View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Croatian Digital Thesis Repository

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Antea Raguž

SUSTAV ZA UPRAVLJANJE BAZOM PODATAKA POSTGRESQL

Diplomski rad

Voditelj rada:
Prof. dr. sc. Robert Manger

Zagreb, Veljača, 2019

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Sadržaj

Uvod

PostgreSQL je relacijska baza podataka sa dugom poviješću. Krajem sedamdesetih godina prošlog stoljeća na Kalifornijskom sveučilištu *Berkeleyju* započinje razvoj relacijskih baza podataka koje su prethodile PostgreSQL bazi. Prva verzija PostgreSQL baze bila je nazvana Ingres. Relational Technologies pretvorile su Ingres u komercijalni proizvod. Relational Technologies stoga je postao Ingres Corporation kojeg je kasnije kupio Computer Associates. Oko 1986. godine Michael Stonebraker sa sveučilišta *Berkeley* vodio je tim koji je dodao objektno-relacijske značajke u jezgru Ingres-a te je nova verzija postala poznata kao Postgres. Postgres je ponovo komercijaliziran, ali ovog put od strane tvrtke Illustra, koja je postala dio Informix Corporation. Andrew Yu i Jolly Chen dodali su SQL podršku Postgresu sredinom 90-ih. Prethodne verzije Postgres-a koristile su drugi, Postgres-specifičan jezik upita poznat kao Postquel. U 1996. godini dodano je puno novih značajki, uključujući MVCC transakcijski model, veća privrženost SQL92 standardu i poboljšanje performansi. Postgres je ponovo promijenio ime, ovog put u PostgreSQL.

Danas je PostgreSQL razvijen od strane međunarodne skupine zagovarača softvera otvorenog koda poznatih kao PostgreSQL Global Development Group. PostgreSQL je proizvod otvorenog koda - besplatan je za korištenje i dostupan je na Windows i Linux platformama. Red Hat je u međuvremenu komercijalizirao PostgreSQL tako što je stvorio Red Hat bazu podataka, ali sam PostgreSQL će i dalje ostati dostupan i besplatan.

Značajke PostgreSQL-a

PostgreSQL je dobro iskoristio svoju dugu povijest te je danas jedna od najnaprednijih dostupnih baze podataka. Navedimo nekoliko značajki koje se nalaze u standardnoj PostgreSQL distribuciji:

- **Objektno-relacijska značajka**: U PostgreSQL-u, svaka tablica definira klasu. PostgreSQL implementira nasljeđivanje između tablica (ili, ako želite, između klasa). Funkcije i operatori su polimorfni.

- **Značajke standarda**: PostgreSQL sintaksa implementira većinu SQL92 standarda i mnogo značajki SQL99. Razlike koje se pojavljuju u sintaksi su najčešće karakteristične za PostgreSQL.

- **Otvoreni kod**: Međunarodni tim programera održava PostgreSQL. Članovi tima dolaze i odlaze, ali glavni članovi ostaju i poboljšavaju PostgreSQL performanse i značajke još od 1996. godine. Prednost PostgreSQL-ove prirode otvorenog koda je ta da se talent i znanje mogu vrbovati prema potrebi. Činjenica da je tim programera internacionalan osigurava da je PostgreSQL proizvod koji se može koristi na bilo kojem jeziku, a ne samo na engleskom.

- **Obrada transakcija**: PostgreSQL štiti podatke i koordinira više istovremenih korisnika kroz cijelu obradu transakcija. Model transakcije koji koristi PostgreSQL temelji se na više-verzijskoj kontroli konkurentnosti (**multi-version concurrency control-MVCC**). MVCC pruža mnogo bolje performanse nego drugi proizvodi koji koordiniraju više korisnika zaključavajući ih na razini tablice, stranice ili retka.

- **Referencijalni integritet**: PostgreSQL implementira cjelovit referencijalni integritet podrža- vanjem odnosa stranog i primarnog ključa kao i okidača, odnosno triggera (SQL procedura koja inicijalizira akciju kada dolazi do događaja `INSERT, DELETE` ili `UPDATE`). Poslovna pravila mogu se izraziti unutar baze podataka umjesto da se oslanjanju na neki od vanjskih alata.

- **Podrška za razne procesorske jezike**: Okidači i drugi procesi mogu biti napisani nekim od procesorskih jezika. Kod napisan da bi se izvršavao na strani poslužitelja najčešće je napisan u PL/pgSQL proceduralnom jeziku sličnom Oracle-ovom PL/SQL. Također kod napisan da bi se izvršavao na strani poslužitelja može se razviti u Tcl-u, Perl-u ili čak u bash-u (Linux/Unix shell-u otvorenog koda).

- **Podrška za više klijentskih API-ja**: PostgreSQL podržava razvoj klijentskih aplikacija na mnogim jezicima poput C, C++, ODBC, Perl, PHP, Tcl/Tk i Python.

- **Jedinstveni tipovi podataka**: PostgreSQL sadrži različite tipove podataka. Osim uobičajenih numeričkih podataka ili stringova, sadrži i geometrijske tipove podataka, Boolean te tipove podatake posebno dizajnirane za rad na mrežnim adresama.

- **Proširivost**: Jedna od najvažnijih značajki PostgreSQL-a jest mogućnost proširenja. Ako ne nađete ono što vam treba, obično to možete dodati sami. Na primjer, moguće je dodati nove tipove podataka, nove funkcije i operatore, čak i nove procese i klijentske jezike. Na internetu postoji mnogo dostupnih paketa, kao na primjer, skup geografskih tipova podataka koji se mogu koristiti za učinkovito modeliranje prostronih podataka, razvijen od tvrtke Refractions Research, Inc.

U prvom poglavlju obrađene su ukratko općenite značajke relacijske baze podataka. Opisan je sustav za upravljanje relacijskom bazom podataka, relacijska algebra te kako modeliramo podatke u relacijskim bazama.

Drugo poglavlje prolazi kroz terminologiju PostgreSQL baze podataka te predstavlja tipove podataka koje koristi PostgreSQL baza.

Kao izvor za pisanje prva dva poglavlja korištena je sljedeća literatura kao izvor [4] [2] [3] [1].

U trećem i zadnjem poglavlju opisana je izrađena aplikacija zasnovana na PostgreSQL sustavu. Opisane su tablice i ostali dijelovi baze podataka.

Poglavlje 1

Relacijske baze podataka

Ovo poglavlje pruža pregled tema vezanih za razvoj baza podataka te razumijevanje njihovog osnovnog koncepata. Poglavlje nije ograničeno samo na objektno-relacijski sustav PostgreSQL, već općenito obuhvaća relacijske baze podataka.

Prvo poglavlje podijeljeno je u tri manja potpoglavlja:

- **Sustav za upravljanje bazom podataka**: Razumijevanje različitih kategorija baza podataka omogućuje programeru snalaženje u svakom sustavu.
- **Relacijska algebra**: Razumijevanje relacijske algebre omogućava jednostavnije savladavanje jezika SQL, a posebno i pisanje SQL koda.
- **Modeliranje podataka**: Pomoću tehnika za modeliranje podataka jednostavnije možemo komunicirati sa drugim korisnicima baza.

1.1 Sustav za upravljanje bazom podataka

Sustavi za upravljanje bazom podataka (DBMS) podržavaju različite scenarije primjene, koriste se u različitim slučajevima te imaju različite zahtjeve. Kako sustavi za upravljenje bazom podataka imaju dugu povijest, prvo ćemo istražiti onu noviju, a zatim kategorije tržišno dominatnih sustava za upravljanje bazom podataka.

Kratka povijest

Baze podataka mogu se definirati kao zbirka ili spremište podataka, koje ima određenu strukturu i kojom upravlja sustav za upravljanje bazom podataka (DBMS). Podaci mogu

biti strukturirani kao tablični, polu-strukturirani kao XML dokumenti ili nestrukturirani podaci koji ne odgovaraju unaprijed definiranom modelu podataka.

U počecima, baze podataka su bile usmjerene na podršku poslovnim aplikacijama, što je dovelo do dobro definirane relacijske algebre i sustava relacijskih baza podataka. Uz uvođenje objekto orijentiranih jezika, pojavile su se nove paradigme upravljanja bazom podataka kao što su objektno-relacijske baze podataka i objektno orijentirane baze podataka.

S pojavom web aplikacija poput društvenih mreža, potrebno je podržati veliki broj zahtjeva na distribuirani način. To je dovelo do druge paradigme baze podataka koja se zove NoSQL (Not Only SQL) koja zahtjeva performansu prije otpornosti na greške i sposobnost horizontalnog skaliranja .

Općenito, na vremenski slijed razvoja baze podataka uvelike su utjecali sljedeći čimbenici:

- **Funkcionalni zahtjevi**: priroda aplikacija koje koriste sustav za upravljanje bazom podataka dovela je do nadogradnje relacijskih baza podataka, poput PostGIS (za prostorne podatke) ili SCI-DB (za znanstvenu analizu podataka).
- **Nefunkcionalni zahtjevi**: uspjesi objektno-orijentiranih programskih jezika stvorili su nove trendove kao što su objektno-orijentirane baze podataka. Objektno-relacijski sustav za upravljanje bazom podataka približio je relacijske baze podataka sa objektno-orijentiranim programskim jezikom. Eksplozija podataka i nužnost obrade velike količine podataka dovela je do stupčastih baza podataka koje se lako mogu zbrajati vodoravno.

Kategorije baze podataka

Mnogi modeli baza podataka pojavili su se i nestali, poput mrežnog i hijerarhijskog modela. Trenutno glavne kategorije baza podataka na tržištu su relacijske, objektno-relacijske i NoSQL baze podataka. NoSQL i SQL baze podataka ne treba doživljavati kao suparničke, ali su međusobno komplementarne. Koristeći različite sustave baza podataka možemo prevladati mnoga ograničenja i dobiti najbolje od različitih tehnologija.

NoSQL baze podataka

Na NoSQL baze podataka utječe CAP teorem, također poznat kao Brewer-ov teorem.

CAP teorem

CAP teorem tvrdi kako je nemoguće za distribuirani računalni sustav istodobno zadovoljiti sva tri sljedeća uvjeta:

- **Konzistentnost**: Svi klijenti (odmah) vide najnovije podatke, čak i u slučaju nadopune.
- **Dostupnost**: Svi klijenti mogu pristupiti podacima, čak i u slučaju pada nekog dijela sustava.
- **Otpornost na particioniranje**: Sustav nastavlja raditi bez obzira na gubitke poruka između mrežnih čvorova.

Izbor izbacivanja nekog od uvjeta određuje prirodu sustava. Na primjer, možemo žrtvovati konzistentnost kako bi postigli skalirani i jednostavni sustav za upravljanje bazom podataka sa visokim performansama.

Konzistentnost je često glavna razika između relacijskih baza podataka i NoSQL baza. Relacijska baza podataka provodi **ACID** što je akronim za **Atomicity, Consistency, Isolation** i **Durability**, odnosno atomičnost, konzistentnost, izolaciju i trajnost. S druge strane, NoSQL baze podatala usvajaju **BASE model** (basically available soft-state, eventual- consistency).

NoSQL motivacija

NoSQL baze podataka su ne-relacijske baze podataka koje mogu pohranjivati podatke, manipulirati njima i pronalaziti ih. NoSQL baze su distribuirane, otvorenog koda i mogu se vodoravno skalirati. NoSQL većinom usvaja BASE model, koji priznaje dostupnost više od dosljednosti. Neformalno jamči da ako niti jedna nova ažuriranja nisu napravljena na podacima, svi podaci koje ispisuje će biti oni najnoviji.

Prednosti NoSQL baza su:

- Jednostavnost dizajna
- Horizontalno skaliranje i jednostavno kopiranje

- Nema shemu
- Podržava ogroman broj podataka

Baze podataka po principu ključ - vrijednost

Pohrana ključa i pridruženih vrijednosti je najjednostavnija pohrana u bazama podataka. U ovom modelu baza podataka skladištenje podataka temelji se na hash tablicama ili mapama. Neki ključevi i pridružene vrijednosti omogućuju pohranjivanje složenih vrijednosti kao liste ili hash tablice. Ključ i pridružene vrijednosti su izuzetno brze za određene scenarije, ali nemaju podršku za složene upite i agregacije. Neki primjeri takvih baza podataka su: Riak, Redis, Memebase i MemcacheDB.

Stupčaste baze podataka

Stupčaste ili stupčasto orijentirane baze podataka temelje se, kao što i ime kaže, na stupcima. Podaci se u pojedinim stupcima dvodimenzionalnih odnosa pohranjuju skupa. Za razliku od relacijskih baza podataka, dodavanje stupaca je jednostavno, a dodaje se pravilom redak-po-redak. Retci mogu imati različit skup stupaca. Tablice imaju koristi od takve strukture tako što se smanjuje memorija za pohranu `NULL` vrijednosti. Ovaj model je najprikladniji za distribuirane baze podataka. HBase je jedna od najpoznatijih stupčastih baza podataka. Temelji se na Google-ovom big table sustavu. Stupčasto orijentirane baze podataka usmjerene su prema stupcima i dizajnirane su za veliku količinu podataka te se jednostavno mogu povećavati. HBase nije prikladan za male baze poodataka.

Baze podataka orijentirane ne dokumente

Baze podataka orijentirane na dokumente prikladne su za polustrukturirane podatke i dokumente. Središnjih koncept takvih baza je pojam dokumenta. Dokumenti spremaju i kodiraju podatke (ili infomracije) u nekom standardnom formatu ili kodu poput XML, JSON i BSON. Dokumenti nisu povezani sa standardnom shemom niti imaju istu strukturu te tako imaju visoki stupanj fleksibilnosti. Za razliku od relacijskih baza podataka, promjena strukture dokumenta je jednostavna i ne zaključava klijentima pristup podacima.

Baze podataka orijentirane na dokumente spajaju snagu relacijskih baza podataka i stupčasto orijentiranih baza. Pružaju podršku ad-hoc upitima i mogu se jednostavno povećavati. MongoDB je primjer jedna takve baze podataka koji učinkovito barata velikom količinom podataka. S druge strane, CouchDB ima visoku razinu dostupnosti čak i u slučaju kvara hardvera.

Baze podataka orijentirane na grafove

Baze podataka orijentirane na grafove temelje se na teoriji grafova gdje se baza podataka sastoji od čvorova i bridova. Čvorovima i bridovima mogu biti pridruženi podaci. Ovakve baze podataka omogućuju prelazak između čvorova preko bridova. Budući da je graf generička struktura podataka, ovakve baze mogu predstavljati različite podatke. Najpoznatija takva baza otvorenog koda je Neo4j.

Relacijske i objektno-relacijske baze podataka

Realacijski sustavi za upravljanje bazom podataka jedan su od najčešće korištenih sustava za upravljanje bazama podataka. Malo je vjerovatno da neka organizacija, institucija ili osobno računalo nema ili ne koristi program koji se oslanja na relacijski sustav.

Mogućnosti relacijskih sustava za upravljanje razlikuju se od jednog proizvođača do drugog, ali većina njih se pridržava ANSI SQL standarda. Relacijske baze podataka su formalno opisane relacijskom algebrom i modelirane relacijskim modelom. **Objektno-relacijska baza podataka (ORD)** slična je relacijskim bazama i podržava objektno orijentirane modele poput: korisničko definiranih i kompleksnih tipova podataka te nasljeđivanje.

ACID postavke

U relacijskoj bazi podataka, jedna logička operacija zove se transakcija. Tehnička realizacija transakcije je skup operacija baza podataka koje su stvaranje, čitanje, ažuriranje i brisanje (CRUD - Create, Read, Update i Delete). U ovom kontekstu ACID svojstva možemo objasniti na sljedeći način:

- **Atomičnost (Atomicity)**: Sve ili ništa, odnosno ako dio transakcije ne uspije, tada se transakcija u cjelini neće izvršiti.
- **Konzistentnost (Consistency)**: Svaka transakcija prebacuje bazu podataka iz jednog važećeg stanju u drugo. Konzistentnost baze podataka regulirana je ograničenjima nad podacima i njihovim međusobnim odnosima.
- **Izolacija (Isolation)**: Istovremeno izvršavanje transakcija u sustavu daje isti rezultat kao da su se transakcije izvršavale serijski.
- **Trajnost (Durability)**: Transakcije se uspješno odrađuju čak i uz gubitak struje ili kod rušenja poslužitelja. To se radi pomoću tehnike nazvane write-ahead log (WAL).

Jezik SQL

Relacijske baze podataka često su povezane s SQL-om (Structured Query Language). SQL je deklarativni programski jezik i standardni jezik relacijskih baza podataka. Američki Nacionalni Standardni Institut (ANSI) i Internacionalna Standardna Organizacija (ISO) prvi put su objavili SQL standard u 1986, nakon čega slijede mnoge verzije kao što su SQL:1999, SQL:2003, SQL:2006, SQL:2008 i tako dalje.

SQL jezik ima nekoliko dijelova:

- **Data definition language (DDL)**: Jezik za definiranje podataka definira i mijenja relacijsku strukturu.
- **Data manipulation language (DML)**: Jezik za rukovanje podacima dovodi i izvlači podatke iz relacija (tablica).
- **Data control language (DCL)**: Jezik za kontrolu nad podacima kontrolira prava pristupa relacijama (tablicama).

Osnovni koncept

Relacijski model predikatna je logika prvog reda, koju je prvi put uveo Edgar F. Codd. Baza podataka prikazana je kao zbirka relacija (tablica). Stanje cijele baze podataka definirano je preko stanja svih relacija u bazi podataka. Različite informacije mogu se izvući iz relacija spajanjem i prikupljanjem podataka iz različitih relacija te primjenom filtera na te podatke.

Nadalje, osnovni pojmovi relacijskog modela uvedeni su odozgo prema dolje pristupom (top-down approach), odnosno prvo opisujemo relacije (tablice), entite, atribute pa domenu. Pojmovi relacija, entitet, atribut i nepoznanica koji se koriste u formalnom relacijskom modelu, jednaki su pojmovima tablica, redak, stupac i `NULL` vrijednost u SQL jeziku.

Relacija

Relaciju možemo zamisliti kao tablicu sa zaglavljem, stupcima i retcima. Naziv tablice i zaglavlje pomažu pri tumačenju podataka u retcima. Svaki redak predstavlja skupinu povezanih podataka koji određeuju neki objekt. Svaka relacija je predstavljena skupom entiteta. Entiteti bi trebali imati isti skup atributa. Atributi imaju domenu, odnosno vrstu i naziv.

Primjer jedne relacije prikazane preko tablice:

id	ime	prezime	e-mail	**telefon**
1	Ante	Antić	ante-antic@gmail.com	1111-111
2	Baro	Barić	baro-baric@gmail.com	2222-222
3	Mato	Matić	mato-matic@gmail.com	NULL

Tablica 1.1: Tablica relacije **osoba**

Shemu relacije predstavlja naziv relacije te imena atributa. Na primjer, osoba (id, ime, prezime, e-mail, telefon) je shema relacije za relaciju osobe. Jedan entitet je jedan redak, a jedan atribut je jedan stupac. Stanje relacije definirano je skupom entiteta. Ako dodajemo, brišemo ili mijenjamo entitete, tada smo promijenili i relaciju. Redoslijed ili položaj entiteta u relaciji nije važan, jer relacija nije osjetljiva na redoslijed. Entitete možemo sortirati preko jednog atributa ili skupa atributa. Bitna činjenica je da ne možemo imati duple retke, odnosno ne možemo imati dva identična entiteta.

Relacija može predstavljati entitete u stvarnom svijetu, kao što je kupac, ili se može koristit za prikaz drugih povezanih relacija. Razdvajanje podataka u raznim odnosima, ključni je koncept u modeliranju relacijskih baza podataka. Taj koncept se naziva normalizacija i on je proces koji organizira odnose stupaca u realcijama radi smanjenja redundancije podataka.

Entitet

Entitet je skup određenih atributa. Pišu se nabrajanjem elemenata unutat zagrada () i odvojenih zarezima, kao na primjer: (Anto, Antić, 29). Elementi entiteta se indetificiraju preko imena atributa. Entiteti imaju sljedeća svojstva:

- $(a_1, a_2, a_3, \ldots a_n) = (b_1, b_2, b_3, \ldots, b_n)$ *ako i samo ako* $a_1 = b_1, a_2 = b_2, \ldots a_n = b_n$
- Entitet nije skup i bitan je raspored atributa:
  1. $(a_1, a_2) \neq (a_2, a_1)$
  2. $(a_1, a_2) \neq (a_1)$
  3. *Entitet ima konačan skup atributa*

U formalnom relacijskom modelu, više-vrijednosni atributi kao ni kompozitni atributi nisu dopušteni. To je važno za smanjenje redundancije podataka i za povećanje dosljed-

njosti podataka. No, ovo nije strogo točno u suvremenim relacijskim sustavima baza podataka zbog korištenja složenih vrsta podataka kao što su JSON. Vlada velika rasprava oko upotrebe normalizacije u takvim slučajevima: u pravilu se normalizacija primjenjuje, osim ako ne postoji dobar razlog da se to ne učini.

Druga bitna stvar su nepoznate vrijednosti, odnosno `NULL` vrijednosti. Predikat u relacijskim bazama podataka koristi tro-vrijednosnu logiku (three-valued logic - 3VL), gdje postoje tri vrijednosne istine: istinito, lažno i nepoznato. U relacijskoj bazi podataka, treća vrijednost, nepoznato, može se tumačiti na više načina, kao što su nepoznati podaci, podaci koji nedostaju ili podaci koji nisu primjenjivi. Tro-vrijednosna logika koristi se za uklanjanje dvosmislenosti.

Atribut

Svaki atribut ima naziv i domenu, a nazivi atributa moraju biti različiti unutar iste relacije. Domena definira mogući skup vrijednosti koje atribut može imati. Jedan od načina definiranja domene atributa jest definiranje tipa podataka i ograničenja tih podataka.

Formalni relacijski model stavlja ogrničenje na domenu koje kaže da bi vrijednost atributa trebala biti nedjeljiva. Na primjer, ako nam je jedan atribut ime osobe koje se sastoji od imena i prezimena te osobe, onda je taj atribut djeljiv na ime osobe i na prezime osobe.

Ograničenja

Relacijski model definira mnoga ograničenja kako bi se kontrolirao integritet podataka, redundancija i valjanost podataka.

- **Redundancija**: Dupli entiteti nisu dozvoljeni
- **Valjanost**: Ograničenja na domeni kontroliraju valjanost podataka.
- **Integritet**: Odnosi unutar jedne baze podataka su međusobno povezani. Ažuriranja ili brisanja entiteta mogu poremetiti odnose unutar relacije.

Ograničenja u relacijskoj bazi možemo podijeliti u dvije kategorije:

- **Naslijeđena ograničenja iz relacijskog modela**: integritet domene, integritet entiteta i ograničenja integriteta relacija

- **Semantička ograničenja, pravila poslovanja i ograničenja specifična za primjenu**: Ova ograničenja se ne mogu eksplicitno izraziti u relacijskom modelu. Međutim, uvođenjem proceduralnog SQL jezika kao što je PL/pgsql za PostgreSQL, relacijske baze podataka također se mogu koristi za modeliranje tih ograničenja.

Ograničenja integriteta domene

Ova ograničenja osiguravaju valjanost podataka. Prvi korak u definiranju ograničenja integriteta domene je određivanje odgovarajućeg tipa podataka. Domena tipa podaka može biti `integer, real, boolean, character, text, inet` i tako dalje. Nakon što navedemo tip podataka, moramo navesti ako postoje i neka ograničenja na taj tip.

- **Ograničenja provjere**: Ograničenja provjere se mogu primjeniti na jedan atribut ili na kombinaciju više atributa unutar entiteta. Pretpostavimo da neki entitet ima atribute `datum_pocetka` i `datum_kraja`. Za te atribute možemo uvesti provjeru (`datum_pocetka` < `datum_kraja`) kako bi osigurali da su datumi ispravno unešeni.
- **Zadana ograničenja**: Atributi mogu imati zadanu vrijednost. Zadana vrijednost može biti fiksna ili može biti dinamična vrijednost koja se temelji na nekoj funkciji poput `random, current time` ili `date`.
- **Ograničenja jedinstvenosti**: Ograničenje jedinstvenosti jamči da atribut ima različite vrijednosti u svakom entitetu. Ono dopušta `NULL` vrijednost. Na primjer, ako imamo tablicu studenata i jedan od atributa je JMBAG, onda moramo osigurati da je ta vrijednost jedinstvena.
- **`NOT NULL` ograničenja**: Prema zadanim postavkama atributa on može poprimati `NULL` vrijednost. `NOT NULL` ograničenje osigurava da atribut ne sadrži `NULL` vrijednost.

Ograničenja integriteta entiteta

U relacijskom modelu, relacija se definira kao skup entiteta. Prema definiciji svaki element skupa je različit, pa tako i svaki entitet u relaciji mora biti različit. Ograničenje integriteta entiteta provodi se nad primarnim ključem koji je atribut ili skup atributa koji imaju sljedeća svojstva:

- Primarni ključ bi trebao biti jedinstven
- Primarni ključ nebi smio poprimati `NULL` vrijednost

Svaka relacija mora imati samo jedan primarni ključ, ali može imati puno jedinstvenih ključeva. Kandidat za ključ je minimalni skup atributa koji mogu indentificirati entitet. Svi jedinstveni, `NOT NULL` atributi mogu biti kandidati za ključ. U praksi, često odabiremo samo jedan atribut za primarni ključ, umjesto više njih kako bi smanjili redundanciju podataka i olakšali međusobno povezivanje relacija.

Ako sustav za upravljanje bazom podataka (DBMS) generira primarni ključ, onda se on naziva zamjenski ključ (surrogat key). Inače se naziva prirodnim ključem (natural key). Kandidati za zamjenski ključ mogu biti sekvence i univerzalni jedinstveni indentifikatori (universal unique identifiers - UUID). Zamjenski ključ ima puno prednosti u izvedbi, prihvaća promjene uvjeta i kompatibilan sa objektnim relacijskim oznakama.

Ograničenja integriteta relacija

Realacije su međusobno povezane zajedničkim atributima. Ograničenja integriteta relacija upravljaju povezivanjem odnosa dvaju ili više relacija te osiguravaju konzistentnost podataka između entiteta. Ako je jedan entitet povezan sa drugim, odnosno ako se jedan entitet referira na drugi entitet, onda taj drugi entitet mora postojati. Nedostatak ograničenja integriteta relacija može dovesti do:

- nevažećih podataka u zajedničkim atributima
- nevažećih informacija tokom spajanja podataka iz različitih relacija

Ograničenja integriteta relacija postižu se pomoću stranih ključeva. Strani ključ je atribut ili skup atributa koji može indetificirati referirani entitet. Budući da je svrha stranog ključa indetificirati entitet u referiranoj relaciji, strani ključevi su općenito primarni ključevi u referiranoj relaciji, odnosno relaciji na koju se povezujemo. Za razliku od primarnog ključa, strani ključ dopušta `NULL` vrijednosti što omogućuje različita modeliranja ograničenja kardinalnosti. Ograničenja kardinalnosti definiraju odnose između dvije relacije. Relacija se može referirati sama na sebe. Takav strani ključ se naziva referentnim ili rekurzivnim stranim ključem. Primjer takve relacije sa referentnim stranim ključem:

id_tvrtke	ime_tvrtke	u_vlasnistvu
1	Facebook	
2	WhatsApp	1

Tablica 1.2: Tablica relacije tvrtka

Atribut `id_tvrtke` je primarni ključ u Tablici 1.2, a atribut `u_vlasnistvu` je strani ključ.

Kako bi osigurali integritet podaka, strani ključevi se mogu koristi za defniranje nekoliko ponašanja kada se referirani entiteti brišu ili ažuriraju. Takva ponašanja nazivamo referentne akcije:

- **Kaskada**: Kada se entiteti brišu ili ažuriraju u referiranoj relaciji, također se brišu ili ažuriraju entiteti u početnoj relaciji.
- **Restrikcija**: Entitet se ne može obrisati ili se referetni atribut ne može ažurirati ako se na njega referira neka druga relacija.
- **Nema akcije**: Slično restrikciji, ali se odgađa do kraja transakcije
- **Zadano**: Kada se briše entitet u referiranom realciji ili se ažurira referirani atribut, stranom se ključu dodijeljuje podrazumijevana vrijednost.
- **Postavljeno na `NULL` vrijednost**: Vrijednost stranog ključa postavljena je na `NULL` vrijednost kada se obriše referirani entitet.

Semantička ograničenja

Semantička ograničenja ili ograničenja poslovne logike opisuju općenita ograničenja baze podataka. Ta ograničenja su provedena od strane poslovne logike ili aplikacijskog programa ili SQL proceduralnog jezika. Prednosti korištenja SQL proceduralnog jezika:

- **Performansa**: Relacijski sustavi za upravljenje bazom podataka (RDBMS) često imaju kompleksne analizatore za generiranje učinkovitih planova izvršenja. U nekim slučajevima rudarenja podataka, količina podatka nad kojom radimo je vrlo velika. Manipuliranje podacima pomoću SQL proceduralnog jezika uklanja mrežni prijenos podataka.
- **Promjene u zadnjoj minuti**: Za SQL proceduralni jezik se mogu implementirati ispravci grešaka bez prekida usluge.

1.2 Relacijska algebra

Relacijska algebra je formalni jezik relacijskog modela. Određuje skup zatvorenih operacija nad relacijama, to jest rezultat svake operacije je nova relacija. Relacijska algebra nasljeđuje mnoge operacije iz skupovne algebre. Relacijska algebra može biti kategorizirana u dvije skupine:

- Prva je skupina operacija koje su naslijeđene iz teorije skupova, kao što su: unije, presjek, skupovna razlika i Kartezijev produkt.
- Druga je skupina operacija koje su specifične za relacijski model poput `SELECT` i `PROJECT`

Operacije relacijske algebre možemo klasificirati kao binarne i unarne operacije. Primitivni operatori su: `SELECT, PROJECT, CARTESIAN PRODUCT, UNION, DIFFERENCE,` i `RENAME`.

Pored primitivnih operatora, postoje funkcije agregacije kao što su `sum, count, min, max` i `average`. Primitivni operatori mogu se koristiti za definiranje ostalih relacijskih operatora poput `left join, right join, join` i `intersection`. Relacijska algebra vrlo je važna zbog izražajne snage u optimizaciji i ponovnom pisanju upita.

SELECT i PROJECT operatori

`SELECT`(σ) je unarni operator čiji je zapis $\sigma_{\varphi}R$ gdje je φ predikat. `SELECT` vraća entitete iz relacije R u kojima je predikat φ istinit.

Dakle, `SELECT` se koristi za restrikciju entiteta iz relacije. Ako nije naveden predikat, tada se vraća cijeli skup entiteta. Na primjer ako želimo osobu čiji je id jednak 2, u relacijskoj algebri imamo zapis:

$$\sigma_{id=2} osoba$$

Gdje koristimo relaciju definiranu u tablici 1.1.

`SELECT` je komutativan operator: upit "daj mi sve osobe kojima je poznat e-mail, a njegovo ime je Ante" možemo napisati na tri načina:

$$\sigma_{e-mail\ is\ not\ null}\ (\sigma_{ime=Ante}\ osoba)$$

$$\sigma_{ime=Ante}\ (\sigma_{e-mail\ is\ not\ null}\ osoba)$$

$$\sigma_{ime=Ante\ AND\ e-mail\ is\ not\ null}\ osoba)$$

Predikate koje odabiremo su određeni tipom podataka. Za numeričke tipove podataka, operatori usporedbe mogu biti: $\neq, =, <, >, \leq, \geq$. Predikatni izrazi mogu sadržavati složene izraze i funkcije.

Ekvivalentni SQL upit relacijskom `SELECT` je `SELECT *`, predikat je definiran u `WHERE` dijelu upita. * znači da tražimo sve atribute iz relacije, no bilo bi dobro izričito navesti koje atribute tražimo.

```
SELECT * FROM osoba WHERE id=2;
```

`PROJECT`(π) je unarna operacija koja se koristi kako bi "odrezala" relaciiju. Zapis te operacije glasi $\pi_{a_1,a_2,...,a_n}R$ gdje je $a_1, a_2, ..., a_n$ skup atributa.

Operacija `PROJECT` mogla bi se vizualizirati kao okomito rezanje tablice. Ako iz tablice želimo samo `ime` i `prezime osobe`, postavimo upit:

$$\pi_{ime,prezime}\ osoba$$

ime	prezime
Ante	Antić
Baro	Barić
Mato	Matić

Tablica 1.3: Rezultat prethodnog upita

Dupliciranih entiteti nisu dopušteni u formalnom relacijskom modelu, stoga je broj vraćenih entiteta operatora `PROJECT` uvijek jednak ili manji od ukupnog broja entiteta u relaciji. Ako se operacija `PROJECT` odnosi na atribut koji je primarni ključ, tada je broj vraćenih entiteta jednak broju entiteta u relaciji.

Ekvivalentni SQL operator za `PROJECT` je `SELECT DISTINCT`. `DISTINCT` je ključna riječ koja se koristi za ukljanjanje dupliciranih vrijednosti. Stoga prethodni upit u SQL jezik zapisujemo:

```
SELECT DISTINCT ime, prezime FROM osoba;
```

Slijed izvršavanja `PROJECT` i `SELECT` operacija može biti zamjenjiv u nekim slučajevima.

Upit "ispište ime osobe čiji je id jednak 2" možemo zapisati i na sljedeće načine:

$$\sigma_{id=2}\ (\pi_{ime,prezime}\ osoba)$$

$$\pi_{ime,prezime}\ (\sigma_{id=2}\ osoba)$$

U drugim slučajevima, operatori `PROJECT` i `SELECT` moraju imati izričit raspored jer ćemo inače dobiti netočan izraz. U slučaju kada tražimo prezime osobe čije je ime Ante, izraz mora biti zapisan u obliku:

$$\pi_{prezime}\ (\sigma_{ime=Ante}\ osoba)$$

`RENAME` operacija

`RENAME`(ρ) je unarna operacija koja koristi atribute. Ona jednostavno preimenuje neki atribut. Ta operacija se većinom koristi kod `JOIN` operacija kako bi mogli razlikovati atribute sa istim imenom iz različitih relacija. Zapis je oblika $\rho_{\frac{a}{b}}R$

Operaciju `RENAME` koristimo, dakle, kada želimo promijeniti naziv nekog atributa ili dati određeno ime rezulatatu neke relacije. Koristimo ju u slučaju kada:

- Želimo ukloniti nejasnoću ako dvije ili više relacija imaju atribute istog naziva
- Želimo osgurati korisnička imena za atribute, osobito ako su povezana sa izvještajima
- Želimo osigurati prikladan način za promjenu definicije relacije, ali još svejedno ostati kompatibilni unatrag

Izraz `AS` je ekvivalentan relacijskoj operaciji `RENAME`. Sljedeći primjer kreira relaciju sa samo jednim entitetom i jednim atributom koji je nazvan `PI`:

```
SELECT 3.14::real AS PI;
```

Skupovne operacije

Skupovne operacije jesu unija, presjek te razlika. Presjek nije osnovni operator relacijske algebre, zato što može biti zapisan pomoću unije i razlike:

$$A \cap B = ((A \cup B) \setminus (A \setminus B)) \setminus (B \setminus A)$$

Unija i presjek su komutativne operacije:

$$A \cup B = B \cup A$$

$$A \cap B = B \cap A$$

`CROSS JOIN` (Karetezijev produkt)

`CARTESIAN PRODUCT` ($\times$) je binarna operacija koja se koristi za generiranje složenijeg odnosa spajanjem svakog entiteta prvog operanda sa svakim entitetom drugog operanda. Pretpostavimo da su R i S ta dva operanda, tada: $R \times S = \{r_1, r_2, ..., r_n, s_1, s_2, ..., s_n\}$, gdje $r_1, r_2, ..., r_n \in R$ i $s_2, ..., s_n \in S$.

`CROSS JOIN` se koristi za spajanje dviju relacija u jednu. Broj atributa u spojenoj relaciji jednak je zbroju broja atributa spojenih relacija. Broj entita u spojenoj relaciji jednak je produktu broja entiteta spojenih relacija.

1.3 Modeliranje podataka

Modeli podataka opisuje entitete u stvarnom svijetu kao što su korisnici, usluge, proizvodi i veze između tih entiteta. Modeli pomažu razvojnim programerima u modeliranju poslovnih zahtjeva i prevođenju tih poslovnih zahtjeva u relacije u relacijskoj bazi podataka.

U poduzećima, modeli podataka igraju vrlo važnu ulogu u postizanju dosljednosti podataka preko interaktivnih sustava. Na primjer, ako entitet nije definiran ili je loše definiran, onda će to dovesti do nedosljednih i pogrešno protumačenih podataka u poduzeću.

Poslovni odjeli u poduzećima nebi trebali definirati poslovna pravila koja opisuju rad na bazama podataka jer to dovodi do kompleksnih struktura podataka. Dakle, poslovni odjeli trebaju definirati što učiniti, ali ne i kako.

Perspektiva modela podataka

Perspektiva modela podataka je definirana od strane ANSI modela:

- **Konceptualni model podataka**: Opisuje semantiku domene i koristi se za komunikaciju o glavnim pravilima poslovanja, akterima i konceptima. Opisuje poslovne zahtjeve na visokoj razini. Konceptualni model je poveznica između programera i poslovnog dijela tvrtke u razvojnom ciklusu aplikacije.

- **Logički model podataka**: Opisuje semantički dio za određenu tehnologiju, kao na primjer, UML class dijagram za objektno orijentirane jezike.
- **Fizički model podataka**: Opisuje kako su podaci zapravo pohranjeni i kako se njima koristi na razini hardvera.

Prema ANSI modelu, ova perspektiva omogućuje promjenu na jednoj razini bez mjenjanja ostalih dijelova. Dakle, možemo mijenjati model na logičkoj i fizičkoj razini bez da ga mjenjamo na konceptualnoj. Na primjer, sortiranje podataka pomoću bubble ili quick sort načina nije od značaja na konceptualnoj razini.

Model entitet-relacija (ER model)

Model entitet-relacija spada u kategoriju konceptualnog modela podataka. ER model se može transformirati u relacijski model uz pomoć određenih tehnika. Konceptualno modeliranje je dio ciklusa razvoja softvera (**Software development life cycle - SDLC**). Konceptualno modeliranje se obično odrađuje nakon funkcionalne faze priklupljanja podataka. U tom trenutku je programer u mogućnosti napraviti prvi nacrt ER dijagrama i opis funkcionalnih zahtjeva pomoću dijagrama toka podataka, sljednog dijagrama, korisničkog scenarija, korisničke priče i ostalih tehnika.

Tijekom faze projektiranja, razvojni programeri baze podataka trebaju posvetiti veliku pozornost na dizajn. Programeri koji modeliraju jednostavne sustave mogu ih isprogramirati direktno. Međutim, treba paziti kod izrade dizajna, jer modeliranje podataka ne uključuje samo algoritme u modeliranju aplikacije nego i same podatke.

Prilikom izrade sheme baze podataka, treba pripaziti na sljedeće zamke:

- **Redundancija podataka**: Loš dizajn baze podataka dovodi do suvišnih podataka. Redundantni podaci mogu uzrokovati razne probleme poput nedosljednosti podataka i degradacije performansi. Prilikom ažuriranja entiteta koji sadrže suvišne podatke, promjena se treba odraziti na sve entitete koji sadrže te podatke.
- **Zasićenje sa `NULL` vrijednostima**: Po prirodi, neke aplikacije imaju rijetke podatke, primjerice u medicini. Zamislimo relaciju koja se zove dijagnoza te ima stotine atributa za simptome neke bolesti poput vrućice, glavobolje, kihanja, kašljanja i tako dalje. Većina ih vrijedi općenito, ali ne i za određenu dijagnozu, pa puno stupaca sadrži `NULL` vrijednosti. Takav slučaj se može modelirati pomoću složenih tipova poput JSON-a, ili pomoću vertikalnog modeliranja entite-atribut-vrijednost (EAV).

- **Čvrsta veza (Tight coupling)**: U nekim slučajevima, čvrsta veza dovodi do složenih i kompliciranih struktura podataka. Budući da se poslovni zahtjevi mjenjaju sa vremenom, neki zahtjevi mogu postati zastarjeli. Modeliranje generalizacije i specijalizacije (na primjer, izvanredni student je student) u čvrstim vezama može izazvati probleme.

Entiteti, atributi i ključevi

ER dijagram predstavlja entitete, atribute i veze. Entitet je prikaz stvarnog objekta kao što je automobil ili osoba. Atribut je svojstvo objekta i opisuje ga, a veza predstavlja odnos ili vezu između dva ili više entieta.

Atributi mogu biti kompozitni ili jednostavni. Kompozitne atribute možemo podijeliti na manje dijelove. Ti manje dijelovi kompozitnih atributa daju nepotpunu informaciju koja sama po sebi semantički nije korisna. Na primjer, adresa se sastoji od imena ulice i kućnog broja, ali samo ime ulice i sami kućni broj nisu adresa.

Atributi mogu biti i jednoznačni i višeznačni. Primjerice, atribut boja ptice je višeznačan jer je ptica rijetko kad jednobojna. Višeznačni atributi mogu imati i gornju i donju granicu koliko vrijednosti dopuštaju. Osim toga, neki se atributi mogu izvesti iz drugih, kao što se dob može izvesti iz datuma rođenja.

Atributi koji su kandidati za ključ mogu identificirati entitet u stvarnom svijetu. Takvi atributi trebaju biti jedinstveni, ali ne moraju nužno biti i primarni ključ, kada fizički modeliramo relaciju. Postoje i složeni atributi koji mogu biti formirani od nekoliko tipova atributa.

Entiteti trebaju imati naziv i skup atributa. Oni su klasificirani na sljedeći način:

- Slabi entiteti: Nemaju svoje atribute koji su kandidati za ključ
- Snažni ili regularni entiteti: Imaju atribute koji su kandidati za ključ

Slabi entiteti su obično povezani sa drugim snažnim, odnosno regularnim entietom. Taj regularni entitet je onda indentificirajući entitet. Slabi entiteti imaju djelomični ključ, tzv. ”diskriminator”. Diskriminator je atribut koji može jedinstveno identificirati slabi entitet i povezan je sa identificirajućim entitetom.

Kada se atribut jednog entiteta referira na drugi entitet, tada postoji neka veza između tih entiteta. U ER modelu te reference ne bi trebale biti modelirane kao atributi, nego kao

veze ili slabi entiteti. Slično kao kod entiteta, postoje dvije klase veza: slabe i snažne. Slabe veze povezuju slabe entitete sa drugim entitetima. Veze mogu imati atribute kao i entiteti.

Veze imaju svojstvo kardinalnosti kako bi se ograničile moguće kombinacije entiteta koje sudjeluju u vezi. Kardinalnost u smjeru prvog i drugog entiteta možemo definirati kao broj primjeraka od drugog entiteta koji mogu biti povezani sa odabranim primjerkom prvog entieta. Kardinalnost navodimo u obliku intervala gornje i donje granice.

Izrada ER dijagrama i relacija

Osnovna pravila za izradu ER dijagrama:

- Povezivanje regularnih entiteta i veza: Ako entiteti imaju kompozitne atribute, onda obuhvaćuje i sve njegove manje dijelove. Za primarni ključ odabiremo jedan od kandidata za ključ.
- Povezivanje slabih entiteta i veza: Uključuje jednostavne atribute i manje dijelove kompozitnih atributa. Dodajemo strani ključ koji se referira na identificirajući entitet. Primarni ključ je tada obično kombinacija parcijalnog ključa i stranog ključa.
- Ako veza ima atribut i funkcionalnost veze je jedan-naprama-jedan (1:1), tada se jedan primjerak iz atributa veze može povezati samo sa jednim primjerkom iz atributa entiteta.
- Ako veza ima atribut i funkcionalnost veze 1:N, tada se jedan primjerak atributa može povezati sa N primjeraka iz vezanog entiteta.
- Ako imamo vezu funkcionalnosti mnogo-naprama-mnogo, dodajemo novi entitet. Dodajemo strani ključ za referenciranje entiteta koji sudjeluju u vezi. Primarni ključ je sastavljen od svih stranih ključeva u vezi.
- Ako imamo više-vrijednosne atribute, dodajemo strani ključ za referiranje na entitet koji posjeduje taj atribut. Primarni ključ se sastoji od stranog ključa i tog više-vrijednosnog atributa.

UML class dijagram

Objedinjeni jezik modeliranja (**Unified modeling language - UML**) je standard razvijen od strane Object Management Group (OMG). UML dijagrami imaju široku primjernu u modeliranju softverskih rješenja te postoji nekoliko tipova UML dijagrama za različite

svrhe modeliranja, uključujući UML class dijagram, use case dijagram, activity dijagram i implementation dijagram.

UML class dijagram može prikazivati atribute, a isto tako i metode. ER dijagram se može jednostavno prevesti u UML class dijagram. UML class dijagram ima nekoliko prednosti:

- **Obrnuto kodiranje**: Shema baze podataka može se lako preokrenuti u generiranje UML class dijagrama.
- **Modeliranje objekata proširene baze podataka**: Moderne relacijske baze podataka imaju nekoliko tipova objekata poput sequence, view, indeksa, funkcija i pohranjenih procedura. UML class dijagrami imaju sposobnost predstavljanja tih tipova.

Poglavlje 2

Upotreba PostgreSQL sustava za upravljanje bazom podataka

2.1 Osnove PostgreSQL-a

PostgreSQL je objektno-relacijska baza podataka koja se temelji na modelu klijent-poslužitelj. Možemo ju usporediti sa glavnim komercijalnim bazama podataka kao što su Sybase, Oracle i DB2. Jedna od glavnih značajki PostgreSQL-a jest ta da je otvorenog koda pa se može vidjeti izvorni kod za PostgreSQL. Iako PostgreSQL razvija organizacija PostgreSQL Global Development Group, on nije u njihovom vlasništvu. Dakle, PostgreSQL je doborovoljno razvijan, održavan i popravljan od grupe programera širom svijeta. PostgreSQL se ne mora kupovati jer je besplatan za korištenje i održavanja, iako se mogu pronaći komercijalni izvori za njegovu tehničku podršku.

PostgreSQL ima sve uobičajene značajke relacijskih baza podataka uz nekoliko jedinstvenih značajki. PostgreSQL podržava hijerarhijsko svojstvo koje je veoma korisno za objektno-orijentirane korisnike. Također, možete dodati svoj tip podataka, a ne samo dodijeliti novo ime postojećem tipu. Uz PostgreSQL možemo dodavati nove osnovne tipove podataka. PostgreSQL podržava geometrijske tipove podataka kao što su točka, linija, poliedar, mnogokut ili kružnica te koristi indeksne strukture koje ubrzavaja te tipove podataka. Nadalje, PostgreSQL se može proširiti izradom novih fukcija, novih operatora i novih tipova podataka u odabranom programskom jeziku. Kako je PostgreSQL temeljen na modelu klijent-poslužitelj, klijentske aplikacije se mogu izgraditi u brojnim programskim jezicima poput: C, C ++, Java, Python, Perl, TCL/Tk... Sa poslužiteljske strane, PostgreSQL podržava vrlo moćan proceduralni jezik PL/pgSQL, ali i sami možete dodavati druge proceduralne jezike poslužitelju. Proceduralni jezici podržavaju Perl, TCL/TK I

bash shell.

Osnovna terminologija baze podataka

S obzirom da PostgreSQL ima dugu povijest i da se puno toga promijenilo u relacijskim bazama podataka od 1977. godine, terminologija se stalno nadograđuje. Puno pojmova koje koristi PostgreSQL ima sinonime sa drugim relacijskim modelima na tržištu. U ovom odjeljku ćemo navesti i objasniti neke od termine koji se koriste u PostgreSQL-u:

- **Shema**

  Shema je imenovana zbirka tablica. Ona također može sadržavati poglede, indekse, sekvence, tipove podataka, operatore i funkcije. Ostale relacijske baze podataka koriste termin katalog. PostgreSQL sustav sadrži shemu pod nazivom `information_schema` koja sadrži skup pogleda i jednu tablicu koji opisuju objekte u trenutnoj bazi. `INFORMATION SCHEMA` automatski postoji u svim bazama u PostgreSQL sustavu. Da bismo pristupili pogledima u toj shemi važno je staviti prefiks `information_schema.ime_pogleda`. Na primjer sljedeći upit daje ime baze podataka u kojoj se nalazimo:

  ```
  SELECT * FROM information_schema.information_schema_catalog_name;
  ```

- **Katalog sustava**

  PostgreSQL ima katalog sustava gdje možemo pronaći podatke specifične za taj sustav koje ne možemo naći u shemi. Katalog sustava PostgreSQL čini skup tablica, a popis tih tablica dan je u `pg_class` tablici:

  ```
  SELECT * FROM pg_class;
  ```

- **PL/pgSQL**

  PL/pgSQL je proceduralno proširenje jezika SQL koje se može koristiti u sustavu PostgreSQL za kreiranje procedura i funkcija. PL/pgSQL podržava kontrolne strukture, jednostavan je za korištenje te nasljeđuje sve (korisnički definirane) tipove, funkcije i operatore iz jezika SQL. Prednost korištenja PL/pgSQL-a je mogućnost grupiranja SQL naredbi te njihovo izvršavanje kao cjelina smanjuje mrežni promet između poslužitelja i klijenta.

- **Baza podataka**

  Baza podataka je imenovana kolekcija shema. Kada se klijentska aplikacija spaja na PostgreSQL poslužitelj, ona sama određuje naziv baze podataka kojoj želi pristupiti. Klijent ne može koristiti više baza podataka po jednom povezivanju, ali može otvoriti više veza kako bi one istovremeno pristupale većem broju baza podataka.

- **Naredba**

  Naredba ili command je string koji šaljete poslužitelju u nadi da će poslužitelj učiniti nešto korisno. Neki ljudi koriste riječ izjava (statement) umjesto naredba.

- **Upit**

  Upit ili query je vrsta naredbe koja vraća podatke od poslužitelja.

- **Tablica**

  Tablica je zbirka redaka koja obično ima ime, ioako tablice mogu biti i privremene, odnosno mogu postojati samo za izvođenje naredbi. Svi retci u tablici imaju isti oblik, odnosno svaki redak u tablici sadrži isti skup stupaca. U drugim sustavima baza podataka koriste se termini poput relacije, datoteka ili klasa (relation, file, class).

- **Stupac**

  Stupac je najmanja jedinica pohrane u relacijskoj bazi podataka. On predstavlja jedan komad informacija o objektu. Svaki stupac ima definirano ime i tip podataka koji se u njemu nalazi. Stupci su grupirani u retke, a retci su grupirani u tablice. Termini poput polja i atributa imaju slično značenje kao stupac tablice.

- **Redak**

  Redak je zbirka vrijednosti stupaca. Kao što je navedeno, svaki redak u tablici ima isti oblik. Ako pokušavate modelirati aplikaciju u stvarnom svijetu, tada redak predstavlja stvarni objekt. Zapis ili entitet su termini koji su ekvivalentni terminu retka.

- **Kompozitni tipovi podataka**

  Mogu se stvoriti novi tipovi podataka koji se sastoje od više vrijednosti. Takve tipove podataka nazivamo kompozitnim tipovima. Na primjer, kada želimo stupac koji prikazuje adresu neke osobe tada ćemo kombinirati string i znamenke.

- **Domena**

  Domena definira imenovanu specijalizaciju drugog tipa podataka, to jest kada isti tip podataka želimo u više različitih tablica.

- **Pogled**

  Pogled ili view je alternativni način predstavljanja jedne ili više tablica. View se može shvatiti kao ”virtualna” tablica. Kada kreirate view više ne spremate podatke, nego samo kreirate drugi način pogleda na postojeće podatke. View je koristan način kako bi se imenovao složeni upit koji ćete morati iznova koristiti.

- **Klijent-poslužitelj**

  Kako je već spomenuto, PostgreSQL je utemeljen na modelu klijent-poslužitelj, odnosno client-server. U klijent-poslužitelj proizvodu, uključena su najmanje dva programa, jedan je klijent, a drugi poslužitelj. Ti programi mogu postojati na istom računalu ili na različitim računalima koji su povezani nekim tipom mreže. U PostgreSQL slučaju, poslužitelj nudi uslugu poput pohrane, obnove i promjene podataka. Klijent šalje upit poslužitelju za obavljanje nekog zadatka: na primjer klijent traži poslužitelja za posluživanje relacijskih podataka.

- **Klijent**

  Klijent je zapravo aplikacija koja šalje zahtjeve PostgreSQL poslužitelju. Prije nego klijentska aplikacija može ostvariti kontakt sa poslužiteljem, mora se spojiti sa **postmasterom** i utvrditi svoj identitet. Klijentske aplikacije pružaju korisničko sučelje i mogu biti napisane različitim programskim jezicima.

- **Poslužitelj**

  PostgreSQL poslužitelj je program koji odrađuje zahtjeve koji su došli od strane klijentske aplikacije. PostgreSQL server nema korisničko sučelje tako da ne možete izravno komunicirati sa poslužiteljom nego samo pomoću klijentske aplikacije.

- **Postmaster**

  Budući da je PostgreSQL klijent-poslužitelj baza podataka, postoji postmaster koji ”sluša” zahtjeve za povezivanjem od klijentske aplikacije. Kada dođe zahtjev za vezu, postmaster stvara novi poslužiteljski proces u operacijskom sustavu poslužiteljkog računala..

- **Transakcija**

  Transakacija je skup operacija baza podataka koje se tretiraju zajedno. PostgreSQL jamči da su izvršene ili sve operacije unutar transakcije ili nije niti jedna. Važno svojstvo transakcije je to da ako dođe do greške u sredini transakcije, operacije prije greške neće biti izvršene. Transakcija obično počinje naradbom `BEGIN` a završava naredbama `COMMIT` ili `ROLLBACK`.

- **`COMMIT`**

  `COMMIT` označava da je transakcija uspješno završila i osigurava da sve operacije napravljene unutar te transakcije budu trajne.

- **ROLLBACK**

  ROLLBACK označava neuspješan završetak transakcije. Kada transakcije završi sa ROLLBACK naredbom, govorite PostgreSQL-u da odbaci sve promjene izvršene na bazi od početka transakcije.

- **Indeks**

  Indeks je struktura podataka koju baza koristi kako bi smanjila količinu vremena koja je potrebna za obavljanje određenih operacija. Indeks također osigurava da se duplicirane vrijednosti ne pojavljuju tamo gdje nisu potrebne.

- **Tablespace**

  Tablespace, odnosno tablični prostor definira alternativno mjesto za pohranu gdje se mogu stvoriti tablice i indeksi. Kada izradite tablicu ili indeks, možete odrediti naziv tabličnog prostora. Ako ne odredite tablični prostor, PostgreSQL stvara sve objekte u istom direktoriju. Tablični prostor se može koristi za raspodjelu opterećenja na više diskovnih pogona. Možemo ga kreirati na sljedeći način:

  ```
  CREATE TABLESPACE tablespace_name OWNER user_name LOCATION
  directory_path;
  ```

  Tablespace-u možemo promijeniti naziv ili vlasnika sljedećim naredbama:

  ```
  ALTER TABLESPACE tablespace_name RENAME TO new_name;
  ```

  ```
  ALTER TABLESPACE tablespace_name OWNER TO new_owner;
  ```

  Nadalje, kao što brišemo tablicu tako možemo izbrisati i tablespace, ali to jedino ako smo njegov vlasnik ili imamo status superusera:

  ```
  DROP TABLESPACE [ IF EXISTS ] tablespace_name
  ```

- **Okidač**

  Okidač ili trigger predstavlja specifikaciju da sustav mora automatski izvršiti određenu funkciju kad god se izvršava određena operacija (naredba) u bazi nad tablicom ili pogledom. Okidač kreiramo na sljedeći način:

  ```
  CREATE TRIGGER trigger_name
     {BEFORE|AFTER|INSTEAD OF}
     {event [OR...]} ON table_name
     [FOR[EACH]{ROW|STATEMENT}]
     EXECUTE PROCEDURE trigger_function;
  ```

  Prije kreiranja okidača potrebno je kreirati funkciju koja se izvršava kao reakcija na nastali događaj. Funkcija koja se koristi unutar okidača `trigger_function` nema

ulaznih argumenata te je povratnog tipa `TRIGGER`. Ista funkcija se može koristiti u nekoliko okidača.

- **Skup rezultata**

  Kada postavite upit u bazu podataka, vratit će se skup rezultata. Skup rezultata sadrži sve retke koji zadovoljavaju postavljeni upit. Skup rezultata može biti i prazan.

2.2 Rad s podacima u PostgreSQL-u

Kada se izradi tablica u PostgreSQL-u, odredi se i tip podataka koja se pohranjuje u svakom stupcu. Na primjer, ako želimo tablicu kupaca i ime kupca, potrebni su abecedni znakovi. Ako spremamo datum rođenja kupca, želimo tip podataka koji se može interpretirati kao datum. Dok će stanje računa biti znamenke.

Svaka vrijednost u PostgreSQL bazi podataka definirana je unutar tipa podataka, a svaki tip podataka ima svoj naziv te niz valjanih vrijednosti. PostgreSQL ima definirane funkcije koje mogu raditi na pojedinom tipu podataka, ali mogu se definirati i nove funkcije. Svaki tip podataka ima skup operatora koji mogu biti korišteni na vrijednostima toga tipa podataka. Operator je simbol koji se koristi za izgradnju složeniji izraza iz jednostavnijih. Neki od aritmetičkih operatora koje već znamo su + (zbrajanje) i − (oduzimanje). Operator predstavlja neku vrstu izračuna koji se primjenjuje na jedan ili više operanada. Na primjer, u izrazu 5 + 3, + je operator, a 5 i 3 su operandi. Operator koji radi na dva operanda zove se *binarni operator*, a operator koji radi na samo jednom operandu se zove *neoperativni operator*.

Kroz ovo poglavlje biti će prikazani tipovi podataka ugrađeni u standardnu PostgreSQL distribuciju. Nadalje, biti će opisani procesi koje PostgreSQL koristi za odluku je li neki operator ili funkcija primjenjiva i ako je, koje vrijednosti zahtjevaju automatsku konverziju tipa podataka.

`NULL` vrijednosti

`NULL` vrijednosti predstavljaju vrijednosti koje nam nedostaju, ili su nam nepoznate ili neprimjenjive. Naravno, prilikom izrade tablice možemo odrediti da određeni stupac ne može sadržavati `NULL` vrijednosti (`NOT NULL`). Time ne utječemo na tip podataka u stupcu, nego samo naglašavamo da `NULL` nije vrijednost koja dolazi u obzir za taj stupac. Stupac koji zabranjuje `NULL` vrijednosti je obavezan, a stupac koji dozvoljava `NULL` vrijednosti nije.

PostgreSQL prepoznaje NULL vrijednosti pregledavajući NULL indikator koji je spremljen odvojeno od stupca. Ako je NULL indikator za zadani redak ili stupac postavljen na TRUE, tada su podaci spremljeni u tom retku ili stupcu beznačajni. To znači da se podaci u retku sastoje od vrijednosti za svaki stupac i niza indikatora od jednog bita za svaki stupac koji daju informaciju dozvoljava li pojedini stupac NULL vrijednost.

Znakovni tip podataka

Postoje tri vrste znakovnog tipa podataka ili stringa koje nudi PostgreSQL. String je niz od nula ili više znakova. Te tri vrste su: CHARACTER(n), CHARACTER VARYING(n) i TEXT.

Tip CHARACTER(n) sadrži niz od n znakova fiksne duljine. Ako u taj tip pohranimo vrijednost kraću od n, tada je vrijednost povećana bjelinama do duljine n. CHARACTER(n) se može skratiti na CHAR(n). Ako izostavimo (n) kod stvaranja stupca CHARACTER, tada se pretpostavlja da je duljina jednaka 1.

CHARACTER VARYING(n) definira niz promjenjive duljine od najviše n znakova. VARCHAR(n) je sinonim za CHARACTER VARYING(n). Ako izostavimo (n) prilikom stvaranja stupca CHARACTER VARYING, možemo pohraniti sve duljine u tom stupcu.

Naposljetku, stupac TEXT odgovara VARCHAR stupcu bez određene duljine - TEXT stupac može pohraniti niz bilo koje duljine.

Numerički tip podataka

PostgreSQL nudi šest numeričkih tipova podataka, od čega su četiri precizne vrijednosti (SMALLINT, INTEGER, BIGINT, NUMERIC(p,s)), a dvije približne (REAL, DOUBLE PRECISION)

Tri od četiri precizna numerička tipa (SMALLINT, INTEGER i BIGINT) mogu pohraniti samo cijele brojeve, dok NUMERIC(p,s) može precizno pohraniti bilo koji broj koji je unutar određenog broja znamenki. (p) predstavlja ukupan broj decimalnih znamenki, a (s) broj decimala.

S druge strane, približne numeričke vrijednosti ne mogu precizno pohraniti sve vrijednosti. Umjesto toga pohranjuju približnu aproksimaciju realnog broja. DOUBLE PRECISION tip podataka može pohraniti ukupno 15 značajnih znamenki, ali kod računanja pomoću DOUBLE PRECISION može doći do pogrešaka u zaokruživanju.

Uobičajeno ime	Sinonim
SMALLINT	INT2
INTEGER	INT, INT4
BIGINT	INT8
NUMERIC(p,s)	DECIMAL(p,s)
REAL	FLOAT, FLOAT4
DOUBLE PRECISION	FLOAT8

Tablica 2.1: Alternativna imena za numeričke tipove podataka

Osim već opisanih numeričkih tipova podataka, PostgreSQL podržava i dva "napredna" numerička tipa: SERIAL i BIGSERIAL. SERIAL je zapravo INTEGER bez predznaka čija se vrijednost automatski povećava (ili smanjuje) kada dodajemo nove retke. Slično, BIGSERIAL je BIGINT sa povećanom vrijednosti. Kada kreiramo stupac koji ima vrijednosti tipa SERIAL ili BIGSERIAL, PostgreSQL automatski kreira SEQUENCE. SEQUENCE je objekt koji generira niz brojeva za korisnika.

Vremenski tip podataka

PostgreSQL podržava četiri osnovna vremenska tipa podataka te nekoliko proširenih tipova koji se bave problematikom vremenskih zona.

DATE se upotrebljava za pohranu datuma, pohranjuje vrijednosti za stoljeće, godinu, mjesec i dan.

TIME je vremenski tip podataka koji pohranjuje vrijednosti za sat, minute, sekunde i mikrosekunde. TIME ne sadrži vremenski zonu, stoga ako želimo uključiti vremensku zonu, upotrebljavamo zapis TIME WITH TIME ZONE ili skraćeno TIMETZ.

Tip podatka TIMESTAMP kombinira prethodna dva DATE i TIME tako što pohranjuje vrijednosti za stoljeće, godinu, mjesec, dan, sat, minute, sekunde i mikrosekunde. Za razliku od TIME, TIMESTAMP uključuje vremensku zonu. Ako ipak želimo samo datum i vrijeme bez vremenske zone, tada koristimo zapis TIMESTAMP WITHOUT TIME ZONE.

Naposljetku, INTERVAL je vremenski tip podataka koji predstavlja raspon vremena. INTERVAL pohranjuje velik broj sekundi koje možemo grupirati u veće jedinice radi lakšeg korištenje i razumijevanja. Sintaksa za INTERVAL omogućuje određivanje broja sekundi u različitim vremenskim jedinicama.

Boolean ili logički tip podataka

PostgreSQL podržava jednu Boolean ili logičku vrstu podataka: `BOOLEAN` ili skraćeno `BOOL`. `BOOLEAN` može sadržavati vrijednosti `TRUE`, `FALSE` ili `NULL` i troši jedan bajt memorije.

Uobičajeno ime	Sinonim
`TRUE`	true, 't', 'y', 'yes', 1
`FALSE`	false, 'f', 'n', 'no', 0

Tablica 2.2: `BOOLEAN` sintaksa

Geometrijski tip podataka

Imamo šest podržavanih geometrijskih tipova podataka koji predstavljaju dvodimenzionalne geometrijske objekte. Osnovni tip je `POINT` koji predstavlja točku unutrar dvodimenzionalne ravnine. `POINT` se sastoji od x i y koordinate, a svaka ta kordinata je numerički tip `DOUBLE PRECISION`.

`LSEG` je geometrijski tip koji predstavlja dvodimenionalnu dužinu. `LSEG` se sastoji od dvije točke tipa `POINT`, jedna koja predstavlja početak, a druga predstavlja kraj dužine.

Geometrijski tip `BOX` koristi se za definiranje pravokutnika - dvije točke koje definiraju `BOX` predstavljaju suprotne vrhove.

`PATH` je skup proizvoljnog broja točaka tipa `POINT` koji su povezani, odnosno `PATH` predstavlja put u dvodimenzionalnoj ravnini. `PATH` može biti otvoren ili zatvoren. Kod zatvorenog puta imamo početnu i krajnju točku koje su povezane, dok kod otvorenog puta one nisu povezane. U PostgreSQL-u podstoje dvije funkcije koje definiraju je li put otvoren ili zatvoren: `POPEN()` i `PCLOSE()`.

`POLYGON` je sličan tip kao zatvoreni put. Razlika između ta dva tipa je u funkcijama koje podržavaju.

Točka koja označava središte tipa `POINT` i točka radijusa tipa `DOUBLE PRECISION` predstavljaju sljedeći geometrijski tip `CIRCLE`.

BLOBs

PostgreSQL sadrži tip podataka koji podržava neobrađene podatke. Neobrađeni podaci su oni čiju strukturu ili značenje vrijednosti baza podataka ne razumije. S druge strane,

PostgreSQL razumije strukturu i značenje drugih tipova podataka, pa tako za tip `INTEGER` razumije da su to cijeli brojevi nad kojima može vršiti zbrajanje, množenje, pretvaranja u nizove i slično. Dok su neobrađeni podaci samo skupina bitova koja nema značenje za PostgreSQL.

`BYTEA` je tip podataka koji služi za pohranu takvih neobrađenih podataka. `BYTEA` je ograničena na pohranu vrijednosti ne većih od 1GB, ali ako trebamo pohraniti veće vrijednosti možemo koristi velike objekte. Veliki objekti se pohranjuju izvan tablice. Na primjer, ako želimo spremiti fotografiju sa svakim redom u tablici, dodali bismo `OID` stupac.

Tip podataka za mrežne adrese

Postoje tri tipa podataka dizajniranih za podržavanje mrežnih adresa u PostgreSQL-u koji podržavaju i IP (Internet Protocol) logičke i MAC (Machine Address Code) fizičke adrese.

`MACADDR` tip je osmišljen za održavanje MAC adrese. MAC adresa je hardverska adresa, obično adresa ethernet sučelja.

`CIDR` sadrži IP adresu mreže i broj značajnih bitova u toj adresi, nazvan netmask.

`INET` može sadržavati IP adresu mreže ili mrežnog računala te broj bitnih bitova u toj adresi, odnosno netmask. Ako je netmask izostavljen, pretpostavlja se da adresa identificira jednog domaćina, odnosno ne postoji vidljiva mrežna komponenta u toj adresi. `INET` vrijednost predstavlja ili mrežu ili host, dok je `CIDR` dizajnira da predstavlja samo adresu mreže.

SEQUENCE

Kod baza podataka, često ćemo imati potrebu pohraniti podatke za koje nemamo niti jednu jedinstvenu oznaku koja bi postala primarni ključ. U tom slučaju ćemo dodijeliti jedinstveni broj svakom entitetu. Kako bismo riješili problem odabira niza takvih jedinstvenih brojeva u PostgreSQL-u postoji objekt koji se naziva `SEQUENCE`. Dakle, `SEQUENCE` je objekt koji automatski generira niz brojeva. Možemo kreirati proizvoljan broj `SEQUENCE` objekata i svakom od njih moramo dodijeliti jedinstveno ime.

Sintaksa za `CREATE SEQUENCE` naredbu:

```
CREATE SEQUENCE name [INCREMENT increment] [MINVALUE min] [MAXVALUE
max] [START start-value] [CACHE cache-count] [CYCLE];
```

Jedino obavezno u naredbi jest ime. Atribut `INCREMENT` određuje iznos za generiranje sljedećeg broja. Ta vrijednost može biti pozitivna ili negativna, ali ne i nula. Pozitivna vrijednost uzrokuje da se brojevi redoslijeda povećavaju, a negativni da se smanjuju. Zadana vrijednost za `INCREMENT` jest 1.

Atributi `MINVALUE` i `MAXVALUE` kontroliraju minimalne i maksimalne vrijednosti za `SEQUENCE`. Kada dođemo do kraja raspona minimalne i maksimalne vrijednosti pomoću atributa `CYCLE SEQUENCE` će biti kružna, odnosno ako je minimalna vrijednost 0 a maksimalna 3, dobijemo ciklus: `0, 1, 2, 3, 0, 1, 2, 3, 0...` Ako ne uključimo atribut `CYCLE` dolazi do poreške: `0, 1, 2, 3, error: reached MAXVALUE`. Zadana vrijednost za `MINVALUE` je -2147483647 i za `MAXVALUE` -1, ako je `INCREMENT` negativan. Ako je `INCREMENT` pozitivan, zadana vrijednost za `MAXVALUE` jest 2147483647, a za `MINVALUE` 1.

Atribut `START` određuje početni broj koji generira `SEQUENCE`. Vrijednost za `START` mora biti između `MINVALUE` i `MAXVALUE`. Zadana vrijednost za `START` jest `MINVALUE`, ako je `INCREMENT` pozitivan, a `MAXVALUE`, ako je `INCREMENT` negativan.

Atribut `CACHE` određuje koliko se slijednih brojeva generira i sprema u memoriju. U većini slučajeva se može koristiti zadana vrijednost.

Postoje tri funkcije koje mogu izvršavati operacije nad `SEQUENCE`. Funkcija `nextval()` stvara/vraća novu vrijednost iz `SEQUENCE`, `currval()` dohvaća najnoviju generiranu vrijednost, dok `setval()` može resetirati `SEQUENCE` na bilo koju vrijednost između `MINVALUE` i `MAXVALUE`.

Nizovi

Jedna od jedinstvenih značajki PostgreSQL-a jest da možemo stupac definirati kao **niz**. Možemo izraditi stupac koji pohranjuje više vrijednosti istog tipa podataka, dok ostale komercijalne baze podataka zahtijevaju da jedan stupac unutar zadanog reda ne može imati više od jedne vrijednosti istog tipa.

Možemo definirati niz bilo kojeg tipa podataka, čak i niz od niza podataka. U tom slučaju smo dobili višedimenzionalni niz, koji se može poistovjetiti sa matricom. Nema ograničenja kod broja članova u nizu, niti kod broja dimenzija niza.

Prikazat ćemo kako definirati dvodimenzionalni niz za ukupno 6 članova:

```
CREATE TABLE arr (pkey serial, val int[2][3]);
```

Redak možemo napuniti vrijednostima na jedan od dva sljedeća načina:

```
INSERT INTO arr( val ) VALUES( '{ {1,2,3}, {4,5,6} }' );
```

ili

```
INSERT INTO arr( val ) VALUES( ARRAY[ [1,2,3], [4,5,6] ] );
```

Indeks za početak niza je zadan na 1. Raspon indeksa niza možemo izmijeniti. Također, elemente niza možemo koristit u bilo kojoj situaciji u kojoj možemo upotrijebiti vrijednosti istog tipa podataka. Na primjer, element niza možemo koristit kod `WHERE` naredbe.

Postoje tri načina kako možemo promijeniti vrijednosti niza. Ako želimo promijeniti sve vrijednosti:

```
UPDATE arr SET val = '{ {7,8,9}, {10,11,12} }' WHERE pkey=1;
```

Ako želimo promijeniti samo jedan element niza:

```
UPDATE arr SET val[2] = 22;
```

Ili ako želimo promijeniti dio niza:

```
UPDATE arr SET val[1:3] = '{11,22,33}';
```

Kada kod kreiranja niza definiramo gornju granicu niza, ona nije fiksna, odnosno ako smo postavili niz na šest elemenata, nema nikakavih ograničenja kod dodavanja sedmog, osmog ili nekog sljedećeg člana niza. To vrijedi samo za jednodimenzionalne nizove.

Nadalje, polje niza može biti `NULL` vrijednost, ali pojedinačni element niza ne može. Odnosno, ne mogu neki elementi niza biti `NULL`, a drugi ne. PostgreSQL će zanemariti naredbu ažuriranja nekog elementa niza na `NULL`. Program neće izbaciti nikakvu grešku, nego jednostavno to ažuriranje neće promijeniti ništa.

Nizovi mogu biti vrlo korisni ako ih se koristi na ispravan način. Trebali bi ih koristit samo onda kada je broj elemenata niza fiksiran nekim stvarnim ograničenjima.

Ograničenja stupaca

Kada kreiramo tablicu u PostgreSQL-u, možemo odrediti i ograničenja stupca. Ograničenje stupca je pravilo koje mora biti zadovoljeno kada unosimo ili ažuriramo vrijednosti u tom stupcu. Također, možemo definirati posebnu tablicu ograničenja koja se odnosi na početnu

tablicu u cjelini, a ne samo na jedan stupac.

Jednom kada definiramo ograničenja stupca, PostgreSQL neće nikada dopustiti da tablica bude u stanju u kojem nisu zadovoljena sva definirana ograničenja. Ako pokušamo unjeti vrijednost koja se kosi sa definiranim ograničenjima, tada ju nećemo moći unjeti. Isto tako ako pokušamo ažurirati tablicu tako da nije u skladu sa ograničenjima, to će ažuriranje biti odbijeno.

Osnovno ograničenje stupca jest: `NULL` i `NOT NULL`. Kao što je već spomenuto, ako stupac ograničimo sa `NOT NULL`, onda on ne smije sadržavati `NULL` vrijednosti. `NULL` ograničenje stupaca zapravo i ne funkcionira kao ograničenje, nego samo kaže da su `NULL` vrijednosti dopuštene u određenom stupcu. Dakle, `NUll` ne prisiljava stupac da sadrži samo `NULL` vrijednosti. Stupac koji je definiran kao `NOT NULL` stupac je obavezan, dok je `NULL` stupac proizvoljan.

`UNIQUE` ograničenje osigurava da stupac sadrži jedinstvene vrijednosti, to jest neće biti dupliciranih vrijednosti u tom stupcu. To ograničenje često koristimo u stupcu koji je definiran kao primarni ključ. Ako pokušamo unjeti dupliciranu vrijednost u `UNIQUE` stupac, primit ćemo poruku o pogrešci. Kada definiramo `UNIQUE` stupac, PostgreSQL će osigurati postojanje indeksa za taj stupac. Ako ga ne stvorimo sami, PostgreSQL će ga samostalno kreirati.

Gotovo svaka tablica koju stvorimo će imati jedan stupac (ili možda više njih) koji jedinstveno indentificira svaki redak. Skup stupaca koji indetificiraju redak naziva se **primarni ključ**. SQL pruža ograničenje, `PRIMARY KEY` koje možemo koristiti za definiranje primarnog ključa za tablicu. Identificiranje stupca (ili skupa stupaca) kao `PRIMARY KEY` je isto kao definiranje stupca koji je `NOT NULL` i `UNIQUE`. No kada stupac definiramo kao `PRIMARY KEY` navodimo da se taj stupac koristi kada trebamo uputiti na pojedini redak u tablici.

Strani ključ (foreign key) je stupac ili grupa stupaca u jednoj tablici koja se odnosi na redak u drugoj tablici. Obično, ali ne uvijek, se strani ključ odnosi na primarni ključ druge tablice. Ograničenje `REFERENCES` govori da se jedna tablica referira na drugu (točnije: strani ključ u jednoj tablici se odnosi na primarni ključ u drugoj tablici). Kreiramo tablicu `table2` čiji se stupac `id` odnosi na primarni ključ tablice `table1`:

```
CREATE TABLE table2 (id CHARACTER(8) REFERENCES table1, number
INTEGER);
```

Nadalje, ograničenje `REFERENCES` ne dopušta promjenu u bazi podataka tako da ono nije zadovoljeno ili je prekršeno. Dakle, ne možemo dodati redak u tablici `table2` koji se

ondosi na nepostojeći redak u tablici `table1`.

Definirajući `CHECK()` ograničenje na stupac, možemo reći da sve vrijednosti umetnute u taj stupac moraju zadovoljiti proizvoljni Booleov izraz. Sintaksa za `CHECK()` je sljedeća:

```
[CONSTRAINT constraint-name] CHECK(boolean-expression)
```

Izvrednjavanje izraza i konverzija tipa

Nakon što smo prošli kroz standardne tipove podataka u PostgreSQL-u, pokazat ćemo kako se mogu kombinirati vrijednosti različitog tipa podataka u složenije izraze.

Kompleksan izraz izrađen je od dva jednostavna izraza i operatora. **Operator** je simbol koji predstavlja neke vrste operacija koje se primjenjuje na jedan ili dva operanda. Kod korištenja operatora, operandi nam mogu biti različite vrijednosti. Na primjer, možemo množiti vrijednosti nekog stupca sa nekom konstantnom vrijednošću. Dakle, možemo kombinirati vrijednosti stupaca, doslovne vrijednosti (konstante), rezultate funkcija i ostale izraze za izgradnju složenih izraza. Ranije je već spomenuto da postoje binarni operatori i neoperativni operatori.

Za neke izraze, posebno one koji kombiniraju različite tipove podataka, PostgreSQL mora obaviti konverziju tipa podataka. Pretvorba tipa koju automatski osigurava PostgreSQL naziva se prisila (coercion). Pretvorba tipa može biti odrađena i od strane programera pomoću operatora `CAST()` ili `::`. Na primjer, ne postoji unaprijed određen operator koji omogućuje zbranje vrijednosti `INT2` tipa i vrijednost `FLOAT8` tipa. PostgreSQL može pretvoriti `INT2` u `FLOAT8` prije zbrajanja, a postoji i operator koji može zbrojiti dvije `FLOAT8` vrijednosti. Svaki računalni jezik definira skup pravila koji upravljaju automtskom pretvorbom tipova podataka, pa tako ni PostgreSQL nije iznimka.

PostgreSQL poprilično jedinstveno podržava korisnički definirane tipove podataka. U većini relacijskih sustava za upravljanje bazama podataka možemo definirati nove tipove podataka, ali to je zapravo samo davanje novih imena postojećim tipovima podataka. U PostgreSQL-u možemo dodati nove tipove podataka koji nisu nužno povezani sa već postojećim tipovima podataka. Kada se definiraju novi tipovi podataka, možemo definirati i nove operatore za rad nad tim novim tipovima. Svaki operator je implementiran kao funkcija operatora, obično zapisana u programskom jeziku C.

Izrada vlastitog tipa podataka

Kreiranje vlastitog tipa podataka u PostgreSQL sustavu nije jedistvena osobina među relacijskim sustavima baza podataka, ali PostgreSQL-a podrška jest jedinstvena. Dok u drugim relacijskim sustavima baza podataka pod nove tipove podataka spadaju već definirani tipovi podataka samo ograničeni određenim vrijednostima, u PostgreSQL sustavu oni spadaju u domenu. PostgreSQL može stvoriti **kompozitne tipove podataka**, odnosno tipove podataka koji sadrže više različitih tipova, Na primjer, adresa: ulica i poštanski broj te grad. Kada se definira kompozitni tip podatka, svaka komponenta ima zasebno ime i tip podataka.

Uz PostgreSQL možemo izraditi i posve nove tipove koji nemaju veze sa već postojećim vrstama. Kada definiramo vlastiti tip podataka, određujemo sintaksu potrebnu za doslovne vrijednosti, format za internu pohranu podataka, skup operatora podržanih za novi tip i skup (unaprijed definiranih) funkcija koje mogu raditi na vrijednostima tog tipa.

Postoje brojni paketi za dodavanje novih tipova podataka u standardnu PostgreSQL distribuciju. Na primjer, PostGIS projekt dodaje geografske tipove podataka bazirane na specifikacijma organizacije Open Geospatial Consortium. PostGIS dodaje nove tipove podataka sa funkcijama, operatorima i indeksima koji se primjenjuju na te prostorne tipove. `/contrib` je direktorij standardne PostgreSQL distribucije koji sadrži tipove podataka o kocki, kao i implementaciju ISBN/ISSN (International Standard Book Number / International Standard Serial Number) tipa podataka.

Profinjenje tipova podataka pomoću `CREATE DOMAIN`

Domena je korisnički definirani tip podataka koji profinjava postojeći tip podataka. Domenu obično stvaramo kada trebamo pohraniti isti tip podataka u puno tablica (ili puno puta unutar iste tablice). Sintaksa za kreiranje domene:

```
CREATE DOMAIN name [AS] data_type [COLLATE collation] [DEFAULT
expression] [CONSTRAINT[ ...  ]  ];
```

Gdje je `CONSTRAINT` izraz oblika:

```
[ CONSTRAINT constraint_name ] {NOT NULL|NULL|CHECK (expression)};
```

Ako u `CREATE DOMAIN` naredbu uključimo `DEFAULT`, tada navedena vrijednost postaje zadana za sve stupce koji imaju isti naziv tipa. Drugim riječima, ako izostavimo stupac `name` u `INSERT` naredbi, PostgreSQL će umetnuti izraz zadan u `expression` umjetno zadane `NULL` vrijednosti. Zadana vrijednost treba zadovoljiti sva ograničenja koja pri-

družujemo domeni.

Nakon što izradimo stupac čiji je tip podataka definiran pomoću domene, tretiramo ga na isti način kao i svaki drugi stupac. Možemo kreirati i indekse koji uključuju vrijednosti domene.

Također, domenu bi bilo dobro definirati i za stupce koji sudjelu u `PRIMARY KEY` i `FOREIGN KEY` odnosu kako bi osigurali da se tipovi podataka tih vrijednosti podudaraju i da tablice mogu uspostaviti vezu jedna između druge pomoću primanog i stranog ključa.

Izrada i upotreba kompozitnog tipa podataka

Kompozitni tip je tip podataka sastavljen od jednog ili više imenovanih polja.

```
CREATE TYPE name_type AS (name1 TYPE1, name2 TYPE2, name3 TYPE3);
```

Dakle, novi kompozitni tip imenovan je sa `name_type` i sastoji se od polja `name1`, `name2` i `name3` koji su `TYPE1, TYPE2`, odnosno `TYPE3` tipa podataka.

Nakon što smo definirali novi tip podataka, možemo stvarati stupce u tablicama toga tipa. Kada dodajemo stupac kompozitnog tipa, dodajemo jedno polje koje se može sastojati od više polja.

```
ALTER TABLE table ADD COLUMN column name_type;
```

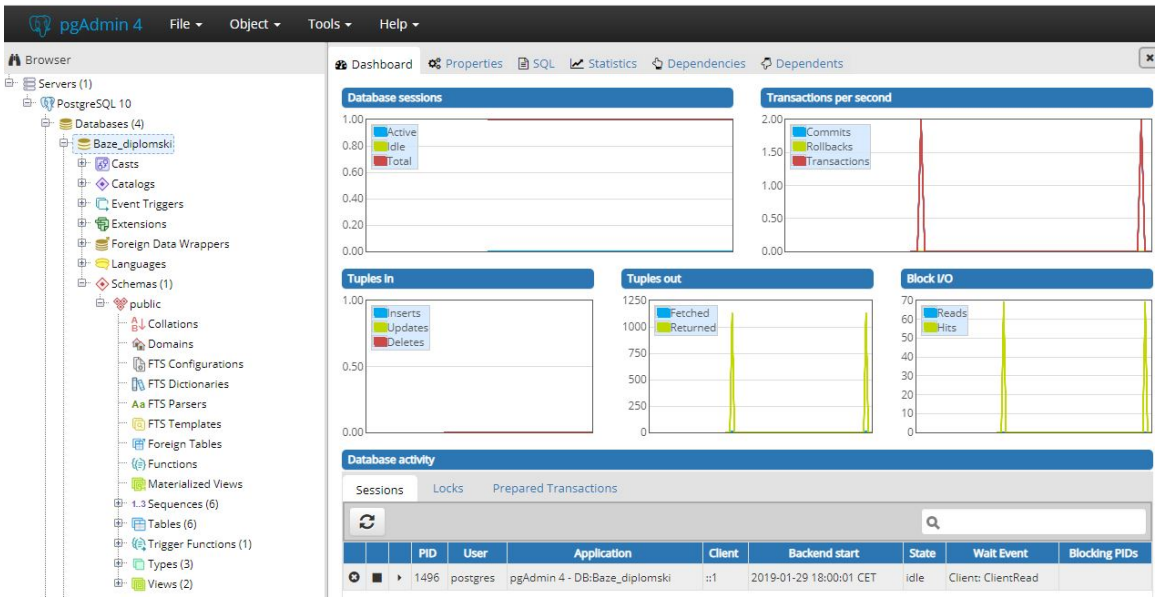
Postoji nekoliko ograničenja koja se tiču kompozitnih tipova podataka. Ne možemo dodati ograničenje kompozitnom tipu, ali možemo priložiti ograničenje domeni i definirati kompozitni tip koji koristi tu domenu. Također, ne možemo stvoriti domenu čiji je tip podataka kompozitni tip, ali možemo stvoriti kompozitni tip koji sadrži domenu. Možemo izraditi ugnježđene kompozitne tipove, odnosno unutar jednog kompozitnog tipa možemo definirati drugi, odnosno više njih, ali takve tipove bolje je izbjegavati radi preglednosti koda. Kada kombiniramo kompozitne tipove i domenu, imamo snažan mehanizam za provođenje ograničenja nad složenim objektima u bazi.

Poglavlje 3

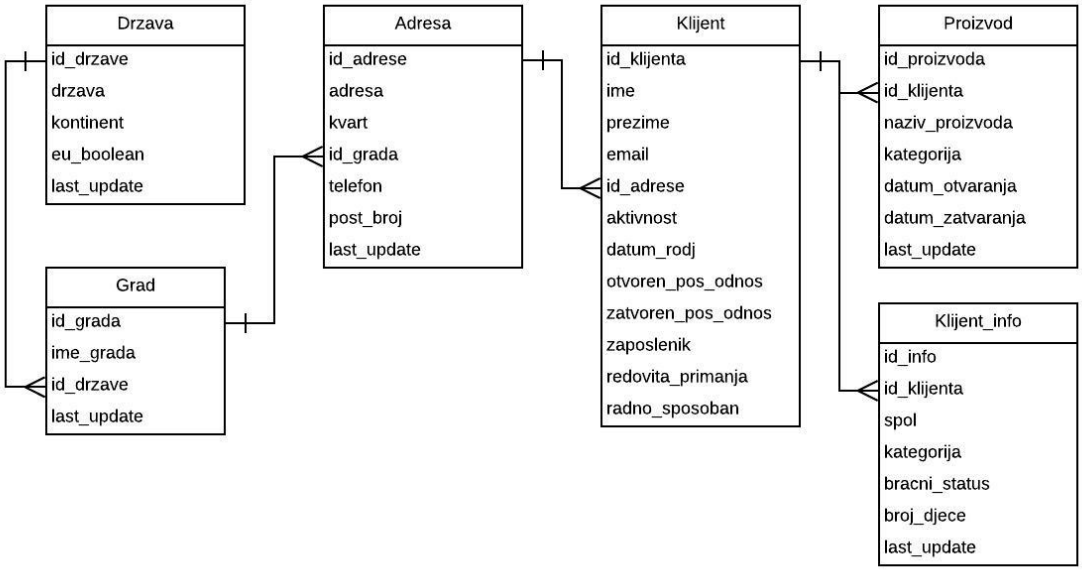
Implementacija baze podataka u pgAdmin-u

PgAdmin je alat za upravljanje PostgreSQL-om, te se može koristit na Linuxu, Unixu, Mac OS X i Windowsu. PgAdmin se može pokrenuti kao web ili kao dekstop aplikacija. Kao alat pogodan je i za korisnike sa slabijim znanjima SQL jezika jer omogućuje jednostavno kreiranje i punjenje tablica bez pisanja upita. Na mjesečnoj bazi se ažurira i popravlja, te se objavljuju nove verzije koje je moguće preuzeti ili ažurirati postojeće. Verzija korištena u ovom radu je pgAdmin 4 v3.2., dok je trenutna najnovija verzija 4 v4.1. (objavljena 15. siječnja 2019).

U pgAdmin alatu kreirana je baza podataka koja se sastoji od: 6 tablica, 1 okidač funkcije, 2 pogleda, 6 sekvence i 3 tipa podataka. Na slici 3.1 prikazan je izgled pgAdmin alata. Dok je na slici 3.2 prikazan ER dijagram tablica. Aplikacija koja je izrađena kao praktični dio ovog diplomskog rada prikazuje klijente Banke, proizvode Banke koje klijent koristi te neke osnovne podatke o klijentima koji su razdvojeni u nekoliko tablica.



Slika 3.1: PgAdmin



Slika 3.2: ER dijagram

Okidač

Kreirana je jedna okidač funkcija `last_update` koja je ugrađena unutar svake tablice, a kod je sljedeći:

```
CREATE FUNCTION public.last_updated()
RETURNS trigger
LANGUAGE 'plpgsql'
COST 100
VOLATILE NOT LEAKPROOF
AS $BODY$

BEGIN
NEW.last_update = CURRENT_TIMESTAMP;
RETURN NEW;
END
$BODY$;
```

Okidač `last_update` zapisuje vrijeme tipa timestamp kada je neki redak unutar tablice nadodan ili izmjenjen.

Sekvence

Za svaku od 6 kreiranih tablica kreirana je sekvenca za primarne ključeve na sljedeći način:

```
CREATE SEQUENCE public.ime_sekvence;
```

Tablice

Jedna od šest tablica navedenih u radu jest tablica **država** koja sadrži podatke o državama, na kojem se kontinentu nalaze te varijablu `eu_boolean` koja označava je li pojedina država članica Europske unije. Primarni ključ jest varijabla `id_drzave` koja je definirana kao sekvenca. Kod glasi:

```
CREATE TABLE public.drzava
(
id_drzave integer NOT NULL DEFAULT nextval('drzava_id_drzave_seq'::regclass),
drzava character varying(50) COLLATE pg_catalog."default" NOT NULL,
kontinent character varying(50) COLLATE pg_catalog."default" NOT NULL,
eu_boolean boolean NOT NULL,
```

```
last_update timestamp without time zone NOT NULL DEFAULT now(),
CONSTRAINT drzava_pkey PRIMARY KEY (id_drzave)
) WITH (OIDS = FALSE)
TABLESPACE pg_default;
```

Također moramo kreirati i okidač funkciju koji koristimo unutar tablice država. Sljedeći kod moramo analogno kreirati i pokrenuti za svaku tablicu koja koristi okidač funkciju `last_update`.

```
CREATE TRIGGER last_updated
BEFORE UPDATE
ON public.drzava
FOR EACH ROW
EXECUTE PROCEDURE public.last_updated();
```

Sljedeća tablica čiji će kod slijediti jest tablica **grad**.

```
CREATE TABLE public.grad
(
id_grada integer NOT NULL DEFAULT nextval('grad_id_grada_seq'::regclass),
ime_grada character varying(50) COLLATE pg_catalog."default" NOT NULL,
id_drzave smallint NOT NULL,
last_update timestamp without time zone NOT NULL DEFAULT now(),
CONSTRAINT grad_pkey PRIMARY KEY (id_grada),
CONSTRAINT fk_grad FOREIGN KEY (id_drzave)
REFERENCES public.drzava (id_drzave) MATCH SIMPLE
ON UPDATE NO ACTION
ON DELETE NO ACTION
) WITH (OIDS = FALSE)
TABLESPACE pg_default;
```

Tablica grad sadrži strani ključ `id_drzave` koji se referira na prethodnu tablicu država. Primarni ključ tablice grad jest `id_grada` te je isto definiran kao sekvenca kao i primarni ključ kod prethodne tablice. Primarni ključevi u sljedećim tablicama su definirani na analogan način.

Adresa je naziv sljedeće tablice:

```
CREATE TABLE public.adresa
(
id_adrese integer NOT NULL DEFAULT nextval('adresa_id_adrese_seq'::regclass),
adresa character varying(50) COLLATE pg_catalog."default" NOT NULL,
kvart character varying(20) COLLATE pg_catalog."default" NOT NULL,
id_grada smallint NOT NULL,
telefon character varying(20) COLLATE pg_catalog."default" NOT NULL,
last_update timestamp without time zone NOT NULL DEFAULT now(),
post_broj integer NOT NULL,
CONSTRAINT adresa_pkey PRIMARY KEY (id_adrese),
CONSTRAINT fk_adresa FOREIGN KEY (id_grada)
ON UPDATE NO ACTION
ON DELETE NO ACTION
) WITH (OIDS = FALSE)
TABLESPACE pg_default;
```

Tablica adresa sadrži podatke o različitim adresama i telefonskim brojevima klijenata. Postoji strani ključ unutar tablice `id_grada` koji se referira na tablicu grad. Na analogan način je primarni ključ tablice adresa `id_adrese` strani ključ u tablici **klijent**, koju ćemo sljedeću opisati:

```
CREATE TABLE public.klijent
(
id_klijenta integer NOT NULL DEFAULT
nextval('klijent_id_klijenta_seq'::regclass),
ime character varying(45) COLLATE pg_catalog."default" NOT NULL,
prezime character varying(45) COLLATE pg_catalog."default" NOT NULL,
email character varying(50) COLLATE pg_catalog."default",
id_adrese smallint NOT NULL,
aktivnost boolean NOT NULL DEFAULT true,
last_update timestamp without time zone DEFAULT now(),
datum_rodj date NOT NULL,
otvoren_pos_odnos date NOT NULL,
zatvoren_pos_odnos date NOT NULL,
zaposlenik boolean,
redovita_primanja boolean,
CONSTRAINT klijent_pkey PRIMARY KEY (id_klijenta),
```

```
CONSTRAINT klijent_id_adrese_fkey FOREIGN KEY (id_adrese)
REFERENCES public.adresa (id_adrese) MATCH SIMPLE
ON UPDATE CASCADE
ON DELETE RESTRICT
) WITH (OIDS = FALSE)
TABLESPACE pg_default;
```

Preostale dvije tablice **klijent_info** i **proizvod** se obje referiraju na tablicu klijent preko stranog ključa `id_klijenta`. Tablica klijent_info sadrži neke dodatne informacije o klijentima, dok su u tablici proizvod popisani proizvodi pojedinog klijenta. Slijede kodovi za te dvije tablice:

```
CREATE TABLE public.proizvod
(
id_proizvoda integer NOT NULL DEFAULT
nextval('proizvod_id_proizvoda_seq'::regclass),
id_klijenta smallint NOT NULL,
last_update timestamp without time zone NOT NULL DEFAULT now(),
naziv_proizvoda character varying(30) COLLATE pg_catalog."default" NOT
NULL,
kategorija character varying(30) COLLATE pg_catalog."default" NOT NULL,
datum_otvaranja date NOT NULL,
datum_zatvaranja date NOT NULL,
CONSTRAINT proizvod_pkey PRIMARY KEY (id_proizvoda),
CONSTRAINT proizvod_id_klijenta_fkey FOREIGN KEY (id_klijenta)
REFERENCES public.klijent (id_klijenta) MATCH SIMPLE
ON UPDATE CASCADE
ON DELETE RESTRICT
) WITH (OIDS = FALSE)
TABLESPACE pg_default;
```

Za posljednju tablicu klijent_info su kreirana tri posebna tipa podataka čiji će kodovi biti navedeni nakon koda za tablicu:

```
CREATE TABLE public.klijent_info
(
id_info integer NOT NULL DEFAULT nextval('klijent_info_id_info_seq'::regclass),
id_klijenta smallint NOT NULL,
spol spol DEFAULT 'M'::spol,
kategorija kategorija DEFAULT 'nepoznato'::kategorija,
```

```
bracni_status brak DEFAULT 'nepoznato'::brak,
last_update timestamp without time zone DEFAULT now(),
broj_djece integer,
CONSTRAINT info_pkey PRIMARY KEY (id_info),
CONSTRAINT info_id_klijenta_fkey FOREIGN KEY (id_klijenta)
ON UPDATE CASCADE
ON DELETE RESTRICT
) WITH (OIDS = FALSE)
TABLESPACE pg_default;
```

Tip podataka

U prethodnoj tablici su korištena 3 novo definirana tipa podataka `spol`, `kategorija` i `brak`. To su jednostavni tipovi podataka definirani na sljedeći način:

```
   CREATE TYPE public.spol AS ENUM ('M', 'Z');

   CREATE TYPE public.kategorija AS ENUM ('maloljetan', 'umirovljenik',
'student', 'zaposlen', 'nezaposlen', 'nepoznato');

   CREATE TYPE public.brak AS ENUM ('u braku', 'nije u braku', 'razveden',
'nepoznato');
```

Pogled

Od tablica koje su navedene u radu su kreirana dva pogleda **klijenti** i **zaposlenici**. Pogled klijenti sadrži sve podatke o klijentima Banke koji su povučeni iz svih 6 tablica:

```
CREATE OR REPLACE VIEW public.klijenti AS
SELECT a.id_klijenta,
a.ime,
a.prezime,
a.email,
a.datum_rodj,
f.spol,
a.redovita_primanja,
e.br_otvorenih_proizvoda,
b.adresa,
b.kvart,
b.telefon,
```

```
b.post_broj,
c.ime_grada,
d.drzava,
d.kontinent,
d.eu_boolean
FROM klijent a
LEFT JOIN adresa b ON a.id_adrese = b.id_adrese
LEFT JOIN grad c ON b.id_grada = c.id_grada
LEFT JOIN drzava d ON c.id_drzave = d.id_drzave
LEFT JOIN ( SELECT proizvod.id_klijenta,
count(DISTINCT proizvod.id_proizvoda) AS br_otvorenih_proizvoda
FROM proizvod
WHERE proizvod.datum_zatvaranja >= now()
GROUP BY proizvod.id_klijenta) e ON a.id_klijenta = e.id_klijenta
LEFT JOIN klijent_info f ON a.id_klijenta = f.id_klijenta
WHERE a.zaposlenik = false OR a.zaposlenik IS NULL;
```

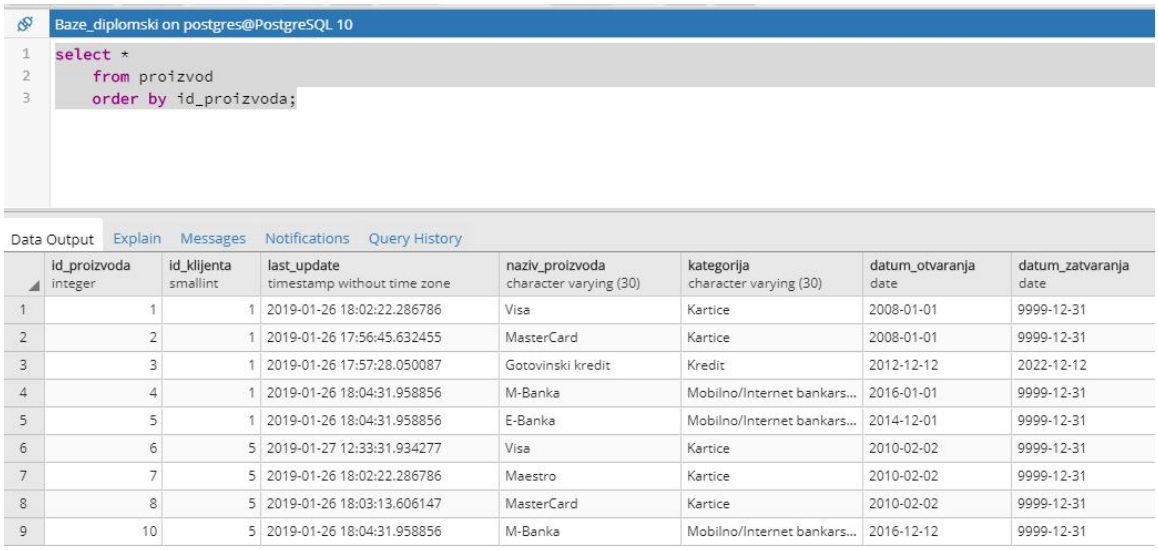
Pogled zaposlenici sadrži podatke o klijentima koji su ujedno i zaposlenici Banke.

```
CREATE OR REPLACE VIEW public.zaposlenici AS
SELECT a.id_klijenta,
a.ime,
a.prezime,
a.email,
a.datum_rodj,
e.spol,
a.redovita_primanja,
b.adresa,
b.kvart,
b.telefon,
b.post_broj,
c.ime_grada,
d.drzava,
d.kontinent,
d.eu_boolean
FROM klijent a
LEFT JOIN adresa b ON a.id_adrese = b.id_adrese
LEFT JOIN grad c ON b.id_grada = c.id_grada
LEFT JOIN drzava d ON c.id_drzave = d.id_drzave
```

```
LEFT JOIN klijent_info e ON a.id_klijenta = e.id_klijenta
WHERE a.zaposlenik = true;
```

3.1 Primjer korištenja opisane baze podataka

Analizirati ćemo tablicu **proizvod**. U poslovnom svijetu će nas često zanimati koji proizvodi su najzastupljeniji, te neke analize nad tim proizvodima i klijentima koji koriste te proizvode.



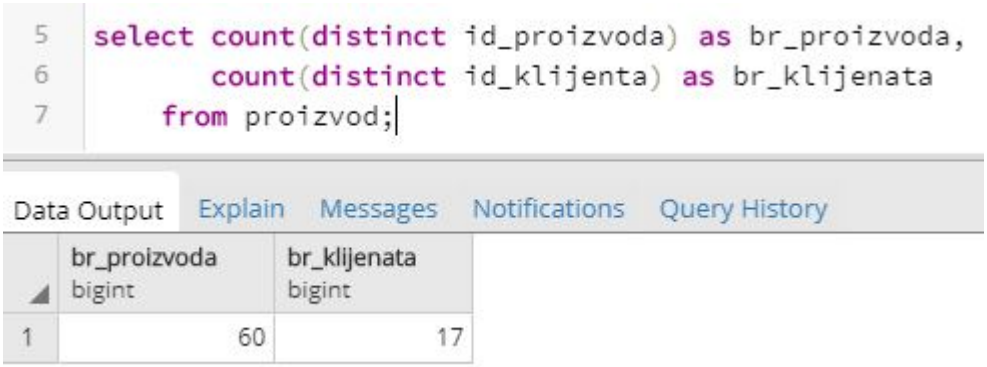
Baze_diplomski on postgres@PostgreSQL 10

```
select *
    from proizvod
    order by id_proizvoda;
```

Data Output | Explain | Messages | Notifications | Query History

	id_proizvoda integer	id_klijenta smallint	last_update timestamp without time zone	naziv_proizvoda character varying (30)	kategorija character varying (30)	datum_otvaranja date	datum_zatvaranja date
1	1	1	2019-01-26 18:02:22.286786	Visa	Kartice	2008-01-01	9999-12-31
2	2	1	2019-01-26 17:56:45.632455	MasterCard	Kartice	2008-01-01	9999-12-31
3	3	1	2019-01-26 17:57:28.050087	Gotovinski kredit	Kredit	2012-12-12	2022-12-12
4	4	1	2019-01-26 18:04:31.958856	M-Banka	Mobilno/Internet bankars...	2016-01-01	9999-12-31
5	5	1	2019-01-26 18:04:31.958856	E-Banka	Mobilno/Internet bankars...	2014-12-01	9999-12-31
6	6	5	2019-01-27 12:33:31.934277	Visa	Kartice	2010-02-02	9999-12-31
7	7	5	2019-01-26 18:02:22.286786	Maestro	Kartice	2010-02-02	9999-12-31
8	8	5	2019-01-26 18:03:13.606147	MasterCard	Kartice	2010-02-02	9999-12-31
9	10	5	2019-01-26 18:04:31.958856	M-Banka	Mobilno/Internet bankars...	2016-12-12	9999-12-31

Slika 3.3: Tablica proizvod



```
5  select count(distinct id_proizvoda) as br_proizvoda,
6         count(distinct id_klijenta) as br_klijenata
7     from proizvod;
```

Data Output | Explain | Messages | Notifications | Query History

	br_proizvoda bigint	br_klijenata bigint
1	60	17

Slika 3.4: Broj proizvoda i klijenata u tablici proizvod

Na slici 3.3 prikazano je nekoliko redaka iz tablice proizvod, a na slici 3.4 prikazano je koliko imamo različitih proizvoda u bazi. Slijedi još nekoliko primjera koristeći neke od tablica koje su prethodno opisane.

```
10  select kategorija,
11         count(distinct id_proizvoda),
12         count(distinct id_klijenta)
13      from proizvod
14          where datum_zatvaranja>now()
15      group by kategorija
16      order by kategorija;
17
```

Data Output | Explain | Messages | Notifications | Query

	kategorija character varying (30)	count bigint	count bigint
1	Kartice	34	13
2	Kredit	5	5
3	Mobilno/Internet bankars...	7	4
4	Račun	1	1
5	Štednja	1	1

Slika 3.5: Kategorija proizvoda

Prethoda slika 3.5 prikazuje koji proizvodi po kategoriji su najzastupljeniji među klijentima. Stavljen je uvjet `where datum_zatvaranja>now()` kako bi osigurali da su prikazani samo trenutno aktivni proizvodi, odnosno proizvodi koji nisu zatvoreni. Dakle, možemo zaključiti da od ukupno 48 otvorenih proizvoda, njih 34 su kartice.

Sljedeća slika 3.6 prikazuje koliko aktivnih vrsta kartica ima u tablici te koliko ih klijenata koristi. Na primjer, dalje bi nas mogle zanimati neke podjele po proizvodima u ovisnosti o spolu klijenta ili o bračnom statusu klijenta ili kategoriji klijenta, odnosno je li klijent zaposlen ili nije.

```
19  select naziv_proizvoda,
20         count(distinct id_proizvoda) as br_proizvoda,
21         count(distinct id_klijenta) as br_klijenata
22      from proizvod
23          where datum_zatvaranja>now()
24              and kategorija='Kartice'
25      group by naziv_proizvoda
26      order by br_proizvoda desc;
27
```

Data Output | Explain | Messages | Notifications | Query History

	naziv_proizvoda character varying (30)	br_proizvoda bigint	br_klijenata bigint
1	Visa	13	13
2	Maestro	12	12
3	MasterCard	9	9

Slika 3.6: Kartice

```
22  select distinct c.kategorija,
23         count(distinct id_proizvoda) as br_proizvoda
24  from (select a.*,
25         b.spol,
26         b.bracni_status,
27         b.kategorija as kat_klijenta
28      from proizvod a
29          left join klijent_info b
30              on a.id_klijenta=b.id_klijenta) c
31      where c.kat_klijenta='zaposlen'
32          and c.datum_zatvaranja>now()
33      group by c.kategorija
34      order by br_proizvoda desc;
```

Data Output | Explain | Messages | Notifications | Query History

	kategorija character varying (30)	br_proizvoda bigint
1	Kartice	19
2	Mobilno/Internet bankars...	6
3	Kredit	5
4	Račun	1
5	Štednja	1

Slika 3.7: Zaposleni klijenti i kategorije proizvoda

Slika 3.7 prikazuje koje kategorije proizvoda imaju klijenti koji su zaposleni (ne nužno u Banci). Na sličan način bi se moglo prikazati koje kategorije proizvoda imaju muški klijenti, a koje ženski, odnosno koje kategorije imaju klijenti ovisno o njihovom bračnom statusu.

Ako nas zanimaju neke analize na klijentima Banke koji nisu zaposlenici, pogled **klijenti** sadrži sve podatke o klijentima. Želimo vidjeti kakva je podjela klijenta u ovisnosti koliko proizvoda imaju u Banci. Podijeliti ćemo broj proizvoda u 3 grupe.

```
61  select a.grupa,
62         count(distinct a.id_klijenta) as br_klijenata
63      from (select id_klijenta,
64         case
65          when br_otvorenih_proizvoda in (1,2) then '1 ili 2 proizvoda'
66          when br_otvorenih_proizvoda>=3 then '3 ili više proizvoda'
67          when br_otvorenih_proizvoda is null then 'nema otvorenih proizvoda'
68         end as grupa
69      from klijenti) a
70      group by a.grupa
71      order by br_klijenata desc;
```

Data Output | Explain | Messages | Notifications | Query History

	grupa text	br_klijenata bigint
1	3 ili više proizvoda	6
2	1 ili 2 proizvoda	4
3	nema otvorenih proizvoda	3

Slika 3.8: Broj proizvoda

Prethodna slika 3.8 pokazuje da najveći broj klijenta ima u Banci otvoreno 3 ili više proizvoda. Promatrani su samo klijenti koji nisu zaposlenici Banke, a takvih ima 13.

Ovo su samo neki od primjera kako bi se mogla koristiti opisana baza podataka. U poslovnom svijetu kako bi lakše prepoznali potrebe klijenata, te analizirali prodaju pojedinih proizvoda često koristimo baze podataka. U velikim firmama i kompanijama ima puno više tablica i podataka nad kojima se vrše analize.

Zaključak

Na prvi pogled rad u PostgreSQL sustavu, te kreiranje tablica i pisanje kodova u pgAdminu nije puno drugačije od pisanja kodova u Oraclu u jeziku MySQL. U suštini je to sve isti jezik koji se razlikuje u nijansama i nekim pravilima. Kao korisnik Oracla u poslovne srvhe, kod pisanja upita u PostgreSQL sustavu nisam primjetila puno razlika. Razliku sam uočila kod pgAdmin alata s kojim sam se prvi put susrela kod pisanja ovoga rada. Pojavili su se neki problemi kod pokretanja programa, s obzirom da sam koristila verziju koja se pokreće kao web aplikacija. Uz malo pretraživanja foruma, pronašla sam rješenje tog problema. Problem koji je program javljao pri pokretanju jest da aplikacijski server ne može biti kontaktiran. Do greške je najvjerojatnije dolazilo jer bi prethodna sesija ostala otvorena ili zapisana pa se nova ne može otvoriti dok se prethodna ne izbriše iz foldera. Sve u svemu nisu prepreke koje bi me odvukle od korištenja pgAdmin alata ili PostgreSQL sustava. Kako sam koristila pgAdmin koji se pokreće kao web aplikacija, neophodna je internet konekcija kako bi se povezali na server.

PostgreSQL sustav vrlo je otporan na pogreške i kvarove jer se temelji na ACID (Atomicity - Consistency - Isolation - Durability) pravilima, dok MySQL ima nešto slabiju ACID prolaznost. Sve u svemu, možemo zaključiti da je PostgreSQL baza sa nešto strožim provjerama i boljom ACID usklađenošću nego MySQL. Također, iz tog razloga PostgreSQL ima nešto bolju ponudu za složenije i zahtjevnije projekte, što je dobar razlog za neke firme za prihvaćanje PostgreSQL-a kao povoljnog, ali moćnog rješenja za njihovo poslovanje. Obje baze se mogu dobro optimizirati ovisno o našim potrebama, ali moglo bi se reći da je MySQL učinkovito rješenje za aplikacije u kojima ima velik broj složenih upita. Koja baza će nam biti bolja, na kraju krajeva dosta ovisi o samom korisniku te o načinu projektiranja same baze. Ako želimo donjeti neki generalizirani zaključak kada koristi koju od tih dviju baza, možemo reći da bi se organizacije češće mogle odlučiti za PostgreSQL sustav zbog podrške standardima, stabilnosti i sigurnosti.

Bibliografija

[1] Ibrar Ahmed, Asif Fayyaz i Amjad Shahzad, *PostGreSQL Developer's Guide*, Packt Publishing, Birmingham, UK, 2015.

[2] Susan Douglas i Korry Douglas, *PostgreSQL, Second Edition*, Sams, 2005.

[3] Salahadin Juba, Achim Vannahme i Andrey Volkov, *Learning PostGreSQL*, Packt Publishing, Birmingham, UK, 2015.

[4] Regina Obe i Leo Hsu, *PostGreSQL Up and Running, 2nd Edition*, O'Reilly Media, Sebastopol CA, USA, 2014.

Sažetak

Rad opisuje objetkno-relacijski sustav za upravljanje bazom podataka PostgreSQL razvijen od strane PostgreSQL Global Development Group. PostgreSQL sustav besplatan je za korištenje i dostupan je Windows i Linux platformama te se često koristi kao alternativa za MySQL. U radu su opisane prednosti i mane PostgreSQL sustava te njegova usporedba sa MySQL sustavom. Praktični dio rada sadrži opis aplikacije izrađene u pgAdmin alatu za upravljanje PostgreSQL-om.

Summary

This thesis describes the PostgreSQL database management system developed by PostgreSQL Global Development Group. PostgreSQL is an open source database that is availabe on Windows and Linux platforms. It is often used as an alternative to MySQL. In this thesis are described the advantages and disadvantages of the PostgreSQL and its comparison with the MySQL. The practical part of the thesis contains the description of the application that was created in the pgAdmin PostgreSQL Management Tool.

Životopis

Rođena sam 4. studenog 1993. u Zagrebu. U rodnom gradu pohađala sam Osnovnu školu Horvati na Knežiji prve tri godine obrazovanja te se od četvrtog razreda osnovne škole prebacujem u Osnovnu školu Vrbani. Godine 2008. upisujem opći smjer X. Gimnazije Ivan Supek u Zagrebu. Kroz odrastanje sam već naginjala ka tehničkim znanostima, no htjela sam istražiti i ostale opcije. Stoga, aktivno pričam dva strana jezika, engleski i njemački te pasivno francuski. Nakon položene mature sa vrlo dobrom ocjenom, 2012. godine upisujem preddiplomski sveučilišni studij *Matematike* na Zagrebačkom *Prirodoslovno-matematičkom fakultetu*. Na kojem u rujnu 2016. godine upisujem diplomski sveučilišni studij *Matematičke statistike*.

Kroz studij sam držala instrukcije iz matematike za osnovnu i srednju školu, te vodila treninge iz kickboxinga. U svibnju 2018. zajedno sa 15 drugih studenata informatičkog ili matematičkog usmjerenja, primljena sam na program stručne prakse *Zaba Future Academy* u Zagrebačkoj banci.